Universidad Zaragoza
1542

Proyecto Fin de Carrera

Gestión de juguetes tangibles activos para una mesa interactiva

Autora

Elisa Ubide Garralda

Directores

Dra. Sandra Baldassarri
Dr. Javier Marco Rubio

Escuela de Ingeniería y Arquitectura

Abril 2013

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

1542 Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza

GIGA Affective Lab

Datos Técnicos

Título:	Gestión de juguetes tangibles activos para una mesa interactiva
Autor:	Elisa Ubide Garralda
DNI:	72816013-Z
Titulación:	Ingeniería Informática
Directora:	Dra. Sandra Baldassarri
Codirector:	Dr. Javier Marco Rubio
Departamento:	Informática e Ingeniería de Sistemas
Centro:	Escuela de Ingeniería y Arquitectura
Universidad:	Universidad de Zaragoza
Fecha:	Abril 2013

Gestión de juguetes tangibles activos para una mesa interactiva

RESUMEN

En el *Affective Lab* del Grupo de Informática Gráfica Avanzada (GIGA) de la Universidad de Zaragoza, se ha desarrollado en los últimos años NIKVision, un prototipo de *tabletop* o superficie horizontal activa. Este prototipo permite crear juegos interactivos, basados en la interacción tangible a través de la manipulación de objetos físicos o juguetes en la superficie del *tabletop*. Para la detección de dichos objetos, NIKVision usa una cámara digital convencional y el *framework* TOYVision, un *software opensource* encargado del análisis de la imagen proveniente de la cámara.

La comunicación entre el *framework* TOYVision y la aplicación de los juegos se realizaba mediante el protocolo TUIO, basado en paquetes UDP, y un *socket* TCP. Dichos protocolos presentaban importantes limitaciones para el desarrollo de juegos tangibles avanzados:

- La comunicación era undireccional: el *framework* TOYVision podía enviar mensajes a la aplicación del juego mientras que el sentido inverso de comunicación no era posible, es decir, la aplicación del juego no podía enviar mensajes al *framework* TOYVision.

- Los mensajes del protocolo TUIO estaban limitados a tres tipos: “objeto añadido en la mesa”, “objeto movido en la mesa” y “objeto eliminado de la mesa”. Juguetes más complejos requerirían mensajes más complejos.

El objetivo de este Proyecto de Fin de Carrera ha sido el de ampliar las posibilidades del *framework* TOYVision para la creación de juegos tangibles avanzados para NIKVision, para lo cual:

- Se han sustituido el protocolo TUIO y el *socket* TCP (unidireccionales) por un solo protocolo basado en XML y que utiliza un *socket* TCP bidireccional. De este modo, los paquetes XML permiten enviar tanto la información enviada anteriormente como nuevos tipos de mensajes para nuevas manipulaciones de los juguetes, para que estos, así mismo, puedan ser controlados por el ordenador (siempre y cuando dichos juguetes dispongan de la electrónica adecuada).

- Se ha creado un nuevo módulo en el *framework* TOYVision para la gestión de sensores y actuadores electrónicos embebidos en los juguetes. Dicha gestión se realiza a través de la plataforma de *hardware* libre Arduino, con la que el *framework* TOYVision se comunica a través de una conexión por un puerto COM. Ello ha requerido la creación y gestión de un nuevo protocolo de comunicación con Arduino y del envío y recepción de información para que la aplicación del juego conozca el estado de los sensores y mande órdenes a los actuadores de los juguetes.

Estas nuevas funcionalidades permiten desarrollar juegos basados en la interacción tangible utilizando juguetes activos.

Agradecimientos

Quiero dar las gracias a mis directores, los Dres. Sandra Baldassarri y Javier Marco, sin cuya ayuda e implicación este proyecto no habría sido posible.

Gracias a mi familia, en especial a mis padres, José Carlos y Mª Ángeles, y a mi hermana, Teresa, por darme siempre ánimos y apoyo.

Y gracias a mis amigos y en especial, a Josean, por todos esos buenos momentos, con los que he descubierto que aquello que dicen de que la mejor época es la universitaria, es cierto.

Índice General

Capítulo 1. Introducción

En este primer capítulo se describe la motivación inicial de este Proyecto Fin de Carrera, su contexto de desarrollo y los objetivos a alcanzar. Por último, se presenta la estructura de este documento, dando a conocer tanto los capítulos que componen la memoria principal como los anexos.

1.1 Motivación inicial

El Grupo de Informática Gráfica Avanzada (GIGA) de la Universidad de Zaragoza ha desarrollado en los últimos años NIKVision, un prototipo *tabletop* o superficie horizontal activa. Este prototipo permite crear juegos interactivos para niños, basados en la interacción tangible[1] a través de la manipulación de juguetes (Figura 1).

Figura 1. Niños jugando con el tabletop NIKVision

Para detectar las manipulaciones de los juguetes (Figura 2.a.1) sobre la mesa, el *tabletop* tiene en su interior una cámara infrarroja (Figura 2.a.2) que actúa como sensor. La imagen captada por la cámara es, por lo tanto, un conjunto de blancos y negros que el sensor envía al sistema (Figura 2.a.3) para que realice su análisis con algoritmos de tratamiento de imagen. Este análisis permite identificar los juguetes colocados en la mesa para así conocer sus manipulaciones realizadas, mostrando el resultado de las mismas sobre la mesa mediante un proyector (Figura 2.a.4). Para facilitar esta identificación, se adhiere a la base de los juguetes un marcador (fiducial) que actúa a modo de código visual fácilmente identificable mediante algoritmos de análisis de imagen (Figura 2.b).

[1] Interacción tangible: nuevo paradigma de interfaz de ordenador, en el que el usuario percibe y controla el contenido digital de una aplicación informática a través de manipulaciones de su entorno físico.

a) **b)**

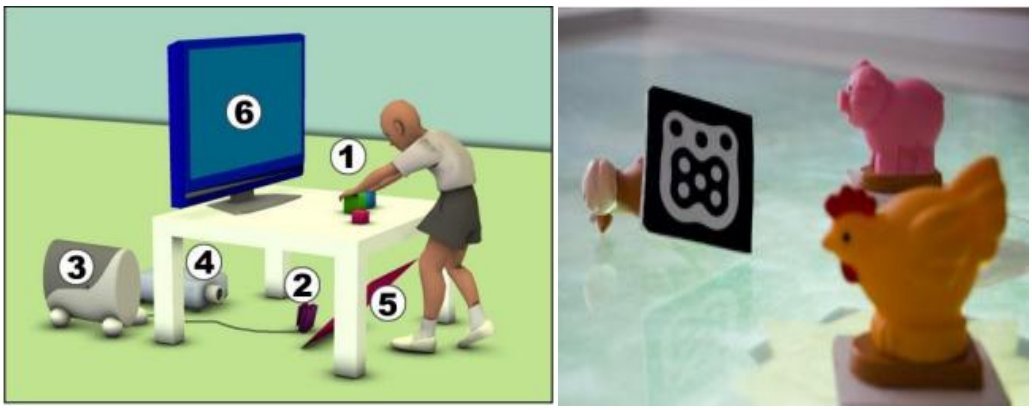


Figura 2. a) Elementos que conforman NIKVision: 1- juguetes pasivos; 2- cámara infrarroja; 3- software de reconocimiento visual (framework TOYVision); 4- las imágenes se proyectan sobre la mesa usando la imagen reflejada sobre un 5- espejo; 6- monitor que muestra el escenario virtual con audio. b) Juguetes con un marcador en su base.

Los fiduciales permiten reconocer si un juguete está sobre la mesa, y si es así, cuál es su posición y su orientación. El niño controla el juego moviendo los juguetes sobre la superficie de la mesa, lo que define a los juguetes como pasivos.

Sin embargo, sería muy interesante que un juguete pudiera, por ejemplo, realizar movimientos o reflejar algún cambio en función de lo que ocurriera en el juego, lo que le convertiría en un juguete activo. Para que un juguete sea activo es necesario embeber en el mismo la electrónica adecuada, a la vez que implementar la comunicación entre el *hardware* de dicha electrónica y el sistema general.

Por lo tanto, desarrollar un juego para el *tabletop* NIKVision con juguetes tanto pasivos como activos implicaría:

1. Analizar la imagen capturada por el sensor (la cámara) para detectar y seguir las manipulaciones de los juguetes.
2. Embeber sensores electrónicos (táctiles, de temperatura…) en los juguetes para poder captar aquellas manipulaciones que la cámara no pueda captar.
3. Embeber actuadores electrónicos (luces, motores…) en los juguetes para que puedan cambiar su estado físico.
4. Comunicar sensores y actuadores electrónicos (analógicos) con el sistema informático (digital) mediante un conversor analógico-digital.

Como se puede ver, desarrollar un juego para NIKVision conllevaría tareas de alta complejidad, muy relacionadas con el *hardware*, lo que ralentizaría el desarrollo, limitándolo a usuarios con alto nivel de conocimientos técnicos.

Para reducir esta complejidad, existen herramientas en el mercado (*toolkits*) que facilitan la labor de desarrollo, aislando el *hardware* y proporcionando información en alto nivel de abstracción.

Por lo tanto, surge la motivación de crear un *toolkit* concreto de desarrollo para poder desarrollar juegos en alto nivel para NIKVision con soporte de juguetes pasivos y activos.

1.2 Contexto de desarrollo

La Figura 3 muestra la arquitectura *software* de TOYVision que actualmente utiliza NIKVision para soportar la manipulación de juguetes.

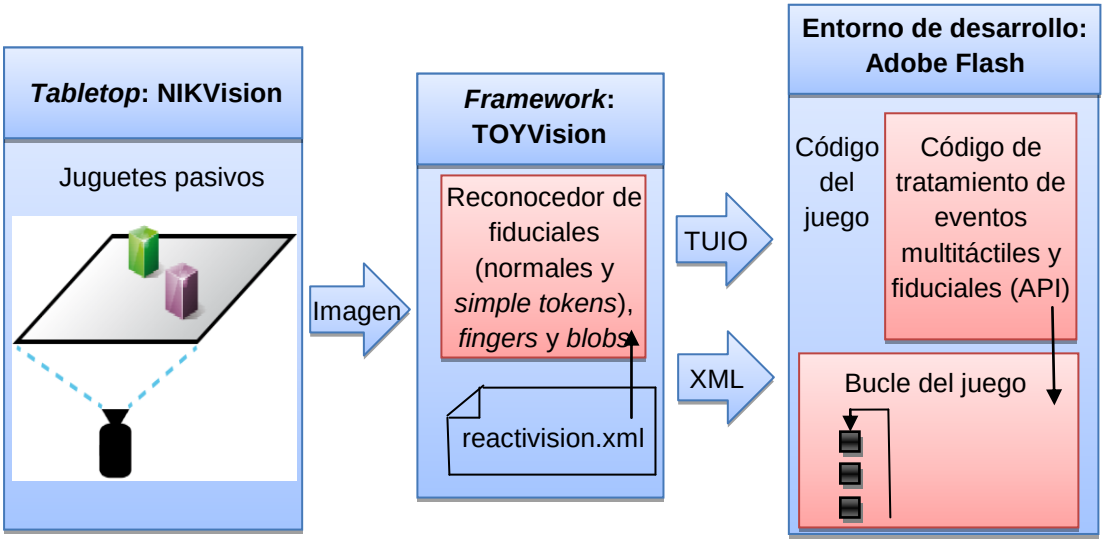


Figura 3. Gráfica de la arquitectura software de TOYVision

A continuación se describen sus componentes:

- El ***tabletop* NIKVision** anteriormente descrito.
- El **entorno de desarrollo** hecho en **Adobe Flash** donde se ejecuta el juego y se encuentra la interfaz de programación de aplicaciones (API). La API se encarga de recibir los mensajes enviados por el *framework* TOYVision para analizarlos y mostrar los resultados correspondientes sobre el *tabletop*. En el caso concreto de NIKVision, dado que los juegos se desarrollan en Action Script 3 de Adobe, se usa la API TUIOAS3 [TUIOAS3 web].
- El ***framework* TOYVision** [TOYVision web], *software* que permite separar el tratamiento de la imagen de la cámara y la detección de objetos, del entorno de desarrollo. Está basado en ReacTIVision [ReacTIVision web] al cual se le han ido añadiendo nuevas funcionalidades en anteriores proyectos fin de carrera de ingeniería informática [Nav11] [Enj10].

Tal y como se puede observar en la Figura 3, cuando el *framework* TOYVision recibe la imagen captada por la cámara, la analiza y envía los eventos correspondientes a la API mediante el protocolo TUIO [TUIO web], [KBBC05] (basado en un protocolo de comunicación UDP, es decir, no orientado a conexión, y mensajes TUIO) o un *socket* TCP para mensajes XML (el canal de comunicación a utilizar depende de la información a enviar). De esta forma, se puede usar cualquier entorno de desarrollo simplemente añadiéndole una API encargada de la gestión de los *sockets* UDP y TCP y del tratamiento de los mensajes TUIO.

Con el *framework* TOYVision se consigue una detección de un conjunto amplio de objetos, pero son únicamente los juguetes pasivos los que se pueden controlar. Es decir, no se pueden utilizar juguetes activos que actúen de forma independiente (iluminándose, moviéndose, emitiendo un pitido…) ni que capten aquello que no pueda captar la cámara (luz, temperatura, presión…) al no tener ni la electrónica ni la comunicación adecuadas.

Los objetos que el *framework* TOYVision es capaz de detectar son fiduciales, *fingers*, *simple tokens* y *blobs* (estos dos últimos objetos fueron añadidos ya que no se detectaban en el ReacTIVision original):

- Un **fiducial** es un conjunto de blancos y negros único, de forma que se le pueda diferenciar de los demás (Figura 4.a).
- Un ***finger*** es un punto blanco, como el que puede ser un dedo de una mano (Figura 4.b).

a) **b)**

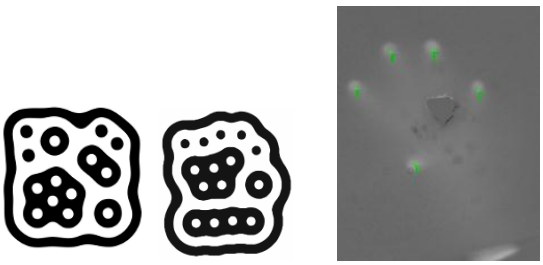


Figura 4. a) Ejemplo de dos fiduciales distintos b) Imagen recibida por ReacTIVision: cinco fingers formados con los dedos de una mano

- Un ***simple token*** es un fiducial simplificado que se puede imprimir en pequeños tamaños para añadirlo a la base de juguetes pequeños (Figura 5), por ejemplo, fichas de juegos de tablero (fichas de la Figura 6.a).
- Un ***blob*** es un objeto no identificado por un fiducial (recortes de cartón blanco de la Figura 6.a). El uso de un *blob* está dirigido a aquellos objetos a los que no se puede o no es aconsejable añadirles un fiducial (manos de los niños, pinceles u otros objetos deformables como plastilina, telas, cartulinas recortables...).

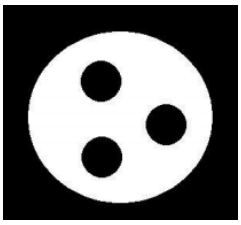


Figura 5. Ejemplo de un simple token

a) **b)**

Figura 6. a) Formas (blobs) y fichas (simple token) creados para jugar. b) Imagen recibida por el framework TOYVision de los dos simple tokens y de los dos blobs de la figura 5.a).

Toda la información relativa a los objetos (fiduciales, *fingers, simple tokens* y *blobs*) se envía al entorno de desarrollo mediante el protocolo TUIO. Sin embargo, dado que los *blobs* tienen una forma aleatoria (elegida por el usuario), se ha de informar de dicha forma al juego. Para ello, y a causa de que el protocolo TUIO solo permite enviar los datos concretos especificados en el ReacTIVision original (en el que solo se detectan fiduciales y *fingers*), se utiliza un *socket* TCP unidireccional (orientado a conexión). A través de este *socket*, se envía un mapa de bits con el contenido de toda la imagen captada por la cámara en formato XML, de tal forma que es posible reconstruir la forma de los *blobs* en el entorno de desarrollo.

1.3 Objetivos del proyecto

En base al contexto planteado en el apartado anterior, el objetivo de este Proyecto Fin de Carrera consiste en ampliar las posibilidades del *framework* TOYVision para la creación de juegos tangibles avanzados que puedan utilizar juguetes activos. Para ello es necesario modificar la arquitectura del *framework* TOYVision (Figura 7) de la siguiente forma:

1- Implementar un protocolo TCP bidireccional que sustituya a los actuales protocolos TUIO y TCP unidireccional, para poder tanto enviar mensajes al juego como recibirlos de él.

2- Sustituir los mensajes TUIO por mensajes XML para poder enviar todo tipo de información, incluyendo la relativa a la electrónica embebida en los juguetes.

3- Crear un nuevo módulo en el *framework* TOYVision para la gestión de sensores y actuadores electrónicos embebidos en los juguetes y conocer así el estado de los sensores y mandar órdenes a los actuadores de los juguetes.

1.4 Estructura de la memoria

Este documento se divide en dos partes: la primera parte contiene la memoria, que resume el trabajo realizado; la segunda contiene los anexos que explican más en detalle ciertos aspectos de la memoria.

La memoria contiene los siguientes capítulos:

- Capítulo 1 - Introducción: describe la motivación inicial del proyecto, su contexto de desarrollo, los objetivos a alcanzar y la estructura de la memoria.
- Capítulo 2 - Análisis del problema: describe las limitaciones que el *framework* TOYVision presenta, un conjunto de requisitos funcionales y no funcionales para hacer frente a esas limitaciones y una propuesta para cumplir estos requisitos.
- Capítulo 3 - Gestor de Juguetes: explica el diseño y la implementación que se ha llevado a cabo para recoger y gestionar toda la información relativa tanto a los juguetes como a los actuadores y/o sensores embebidos en ellos.
- Capítulo 4 - Gestor de XMLSocket: presenta el análisis del nuevo protocolo de comunicación creado para conseguir la bidireccionalidad basada en mensajes XML y su implementación.
- Capítulo 5 - Gestor de Arduino: presenta el diseño realizado para dar solución a la problemática de gestionar diferentes sensores y actuadores embebidos en los juguetes, utilizando para ello una placa Arduino, y describe la programación de la placa Arduino y la comunicación entre el *framework* TOYVision y la placa.
- Capítulo 6 - Resultados obtenidos: presenta el resultado obtenido mediante un caso real como ejemplo.
- Capítulo 7 - Conclusiones: se resumen la consecución de objetivos, el trabajo futuro a realizar y la valoración personal.

Los anexos son los siguientes:

- Anexo A - Documentación del desarrollo software: incluye la metodología de análisis, el fichero de configuración *toys.xml*, la configuración de mensajes y las herramientas utilizadas.
- Anexo B - Arduino: describe cómo es y cómo utilizar una placa Arduino UNO, así como la descripción detallada del uso que se le ha dado en este proyecto.
- Anexo C - Mejoras en el código: se describen las mejoras hechas en el código para una mejor ejecución del *framework* TOYVision.
- Anexo D - Gestión del Proyecto: describe la dedicación en tiempo del proyecto, presentando la distribución de ese tiempo en las distintas tareas.

Capítulo 2. Análisis del problema

En este capítulo se describen en un primer apartado las limitaciones que presenta el *framework* TOYVision, en un segundo apartado los requisitos funcionales y no funcionales necesarios para hacer frente a esas limitaciones, y en un tercer y último apartado los métodos ideados para cumplir dichos requisitos.

2.1 Limitaciones del *framework* TOYVision

Profundizando en el contexto de desarrollo presentado en el capítulo anterior, cuando el *framework* TOYVision recibe la imagen captada por la cámara, la analiza obteniendo la siguiente información:

- Fiduciales:
  - Identificador: se obtiene en función de sus blancos y negros.
  - Número de sesión: único para cada objeto identificado en la superficie de la mesa (lo asigna el *framework* automáticamente). De este modo, si se tienen dos fiduciales cuyo identificador sea el mismo, se podrá diferenciar uno de otro.
  - Posición en el eje x.
  - Posición en el eje y.
  - Orientación.

- *Fingers*: número de sesión, posición en el eje x y posición en el eje y.

- *Simple tokens* (fiduciales simplificados): identificador, número de sesión, posición en el eje x, posición en el eje y, y orientación.

- *Blobs*: número de sesión, posición en el eje x, posición en el eje y, orientación, área y forma.

Tal y como se ha comentado en el capítulo anterior, los *simple tokens* y los *blobs* son los nuevos objetos añadidos en el *framework* TOYVision. Casi todos los datos de estos dos nuevos objetos se pueden enviar a través del protocolo TUIO, ya que coinciden con alguno de los datos calculados para los fiduciales o para los *fingers* (lo que implica que ya se calculaban en el ReacTIVision original y que se enviaban por dicho protocolo). El único valor que es diferente es la forma de los *blobs*, por lo que dada la rigidez del protocolo TUIO para enviar datos distintos a los impuestos, se envía mediante el *socket* TCP en un mensaje XML.

Resumiendo las capacidades hasta ahora mencionadas, el *framework* TOYVision utilizado para la gestión de la imagen recibida desde la cámara es capaz de:

1- Calcular los datos relativos a los fiduciales, *fingers*, *simple tokens* y *blobs*.

2- Usar el protocolo TUIO para enviar aquellos datos que ya se trataran en el *framework* ReacTIVision original.

3- Usar el protocolo basado en un *socket* TCP unidireccional para enviar los datos de aquellos objetos no reconocidos por un fiducial o un *finger*.

En base a lo expuesto, se comprueba la limitación en el *framework* TOYVision para dar soporte a los juguetes activos, ya que: ni el protocolo TUIO ni el nuevo *socket* TCP permiten bidireccionalidad y, por lo tanto, no es posible recibir mensajes desde el entorno de desarrollo.

2.2 Análisis de requisitos

Conociendo las limitaciones latentes en el *framework* TOYVision se pretende con este proyecto superar dichas limitaciones para poder obtener los objetivos fijados.

Tras el análisis del problema (detallado en el anexo A) se obtienen los siguientes requisitos funcionales y no funcionales.

Requisitos funcionales:

RF1- Crear una especificación en lenguaje XML para informar de todas las manipulaciones que recoge el *framework* TOYVision: adición, modificación y eliminación de los fiduciales (normales y *simple tokens*), *fingers* y *blobs*, y uso de sensores y actuadores.

RF2- Implementar un *socket* TCP bidireccional que comunique el *framework* TOYVision y la API.

RF3- Implementar una conexión bidireccional para la comunicación entre el *framework* TOYVision y los sensores y actuadores embebidos en los juguetes.

RF4- Gestionar la información intercambiada entre el *framework* TOYVision y los sensores y actuadores.

RF5- Programar la placa Arduino para el control de los actuadores y sensores embebidos en los juguetes.

RF6- Gestionar la información relativa a los juguetes:

- Diseñar un fichero de configuración que detalle la información relativa a los juguetes.
- Guardar y gestionar la información relativa a los juguetes contenida en el fichero de configuración.

RF7- Filtrar eventos falsos debidos al ruido de la cámara para aumentar la robustez de la detección de las manipulaciones de los juguetes.

Requisitos no funcionales:

RNF1- Modificar el código del *framework* TOYVision existente en lenguaje C++.

RNF2- Usar el entorno de programación Microsoft Visual Studio 2005.

RNF3- Utilizar una placa Arduino UNO para el control de los actuadores y sensores.

RNF4- Utilizar el entorno de programación Arduino 1.0.1 para Windows [Arduino 1.0.1 web] para implementar y cargar el código en la placa Arduino.

RNF5- Imponer que el fichero de configuración relativo a los juguetes sea de tipo XML y almacenarlo en la misma carpeta donde se halle el ejecutable.

2.3 Propuesta de diseño al *framework* TOYVision

Mediante la nueva propuesta de arquitectura *software* de TOYVision presentada en la Figura 7, se propone solucionar las limitaciones presentadas en el primer apartado de este capítulo, tanto modificando parte de lo existente como añadiendo nuevos módulos y protocolos de comunicación (todo aquello en verde indica lo que es responsabilidad de este proyecto). A continuación se describen brevemente los aspectos más importantes de la arquitectura propuesta.

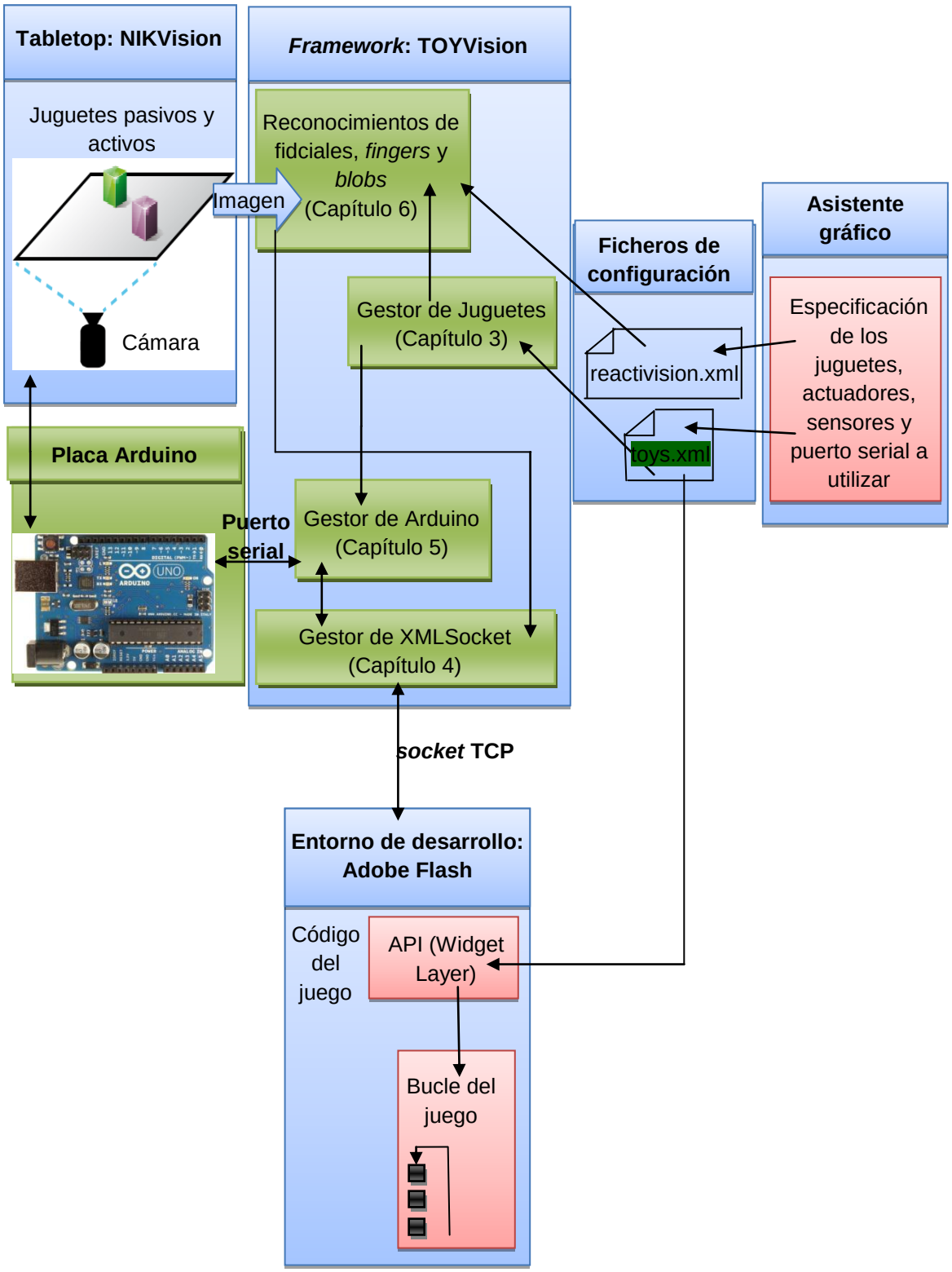


Figura 7. Gráfica de la propuesta de la nueva arquitectura software de TOYVision

- **Gestor de Juguetes:** se encarga de leer y almacenar el contenido del fichero de configuración ***toys.xml*** (fichero que también utiliza el entorno de desarrollo). Este fichero de configuración lo genera el desarrollador del juego mediante el asistente gráfico [TOYVision web] y contiene toda la información relativa a los juguetes. Una vez hecha la lectura de los datos de los juguetes, tanto el **Gestor de Arduino** como el **reconocimiento de objetos** harán uso de esos datos para sus respectivas funciones.

- **Reconocimiento de fiduciales, *fingers* y *blobs*:** trata la imagen recibida desde el *tabletop* obteniendo y enviando al entorno de desarrollo la información relativa a los fiduciales, *fingers* y *blobs*. Dicho envío se hace mediante el protocolo gestionado por el **Gestor de XMLSocket** donde los mensajes enviados están en lenguaje XML[2]. Esto en nuestro caso se traduce en que el tipo de información a incluir en un mensaje sea totalmente opcional e ilimitado, eliminando la restricción impuesta por el protocolo TUIO.

- **Gestor de XMLSocket:** crea la conexión bidireccional entre el *framework* TOYVision y el entorno de desarrollo mediante un ***socket* TCP bidireccional**. Una vez hecha la conexión, controla dicho *socket* encargándose de los envíos y recepciones de mensajes, y del cierre de la conexión. En el caso de recibir un mensaje desde el *socket*, lo reenvía al **Gestor de Arduino**.

- **Gestor de Arduino:** crea una conexión bidireccional con la **placa Arduino**, que actúa como conversor analógico-digital y a la que están conectados los dispositivos electrónicos (sensores y actuadores) embebidos en los **juguetes activos**. Una vez hecha la conexión, se encarga de:
    - Enviar a la **API** a través del **Gestor de XMLSocket** lo captado por los sensores.
    - Transmitir a la **placa Arduino** las órdenes, previamente recibidas por el **Gestor de XMLSocket**, a ejecutar sobre los actuadores.

Además de las nuevas funcionalidades es necesario revisar y modificar el código del *framework* TOYVision existente para lograr una mayor estabilidad y robustez a la detección de fiduciales y *blobs*. En el Anexo C se presentan en detalle los problemas de estabilidad que se han ido encontrando a lo largo del proyecto y las soluciones que se han aplicado para solucionar los mismos.

[2] Lenguaje XML: lenguaje de marcas desarrollado por World Wide Web Consortium (W3C) [W3C web] que permite definir la gramática de lenguajes específicos.

Capítulo 3. Gestor de Juguetes

En este tercer capítulo se explica el diseño y la implementación que se ha llevado a cabo para recoger y almacenar toda la información relativa a los juguetes, así como la de los actuadores (digitales) y/o sensores (analógicos o digitales), es decir, los dispositivos electrónicos, embebidos en ellos.

El análisis completo de este Gestor de Juguetes se muestra de forma detallada en el Anexo A (sección A.1) de este documento, donde además se puede ver su relación con el resto de las partes del proyecto.

3.1 Diseño del Gestor de Juguetes

Para poder tener un mayor número de recursos disponibles y así ampliar el tipo de juegos a crear, se planteaba como objetivo poder utilizar juguetes activos además de los juguetes pasivos ya existentes.

Un juguete se denomina activo cuando puede actuar de forma independiente o puede detectar por sí mismo aquello que la cámara del *tabletop* no detecte, para lo cual ha de tener embebido al menos un actuador o un sensor. Ese dispositivo electrónico es controlado por un conversor analógico-digital que debe estar conectado al ordenador para poder disponer de una comunicación con el *framework* TOYVision. En este proyecto se decidió utilizar una placa Arduino UNO (descrita en detalle en el Anexo B) como conversor analógico-digital, tal y como se explicará en detalle en el Capítulo 5 de esta memoria.

Para poder controlar los juguetes desde el *framework* TOYVision, es necesario conocer la información relativa a la placa Arduino y a los juguetes que se van a utilizar, como se detalla a continuación (todos los datos que siguen se presentan con más detalle en el Anexo A, sección A.2):

- Arduino
    - Identificador del Arduino.
    - Puerto COM del ordenador al que está conectado el Arduino.

- Juguetes
    - Nombre del juguete.
    - Identificador del fiducial de la base del juguete.
    - Dispositivos electrónicos asociados (si es el caso)
        - Nombre del dispositivo electrónico.
        - Terminal de la placa Arduino al que el dispositivo electrónico está conectado.
        - Identificador de la placa Arduino a cuyo terminal está conectado el dispositivo electrónico.

- Tipo del dispositivo electrónico:
  - ‘non_continuous’ (servo de 180º).
  - ‘continuous’ (servo de 360º).
  - ‘switch’ (el valor relativo al dispositivo electrónico es ‘0’ o ‘1’).
  - ‘analog’ (el valor relativo al dispositivo electrónico está entre 0 y 1023, ambos inclusive).
- Comandos:
  - Nombre del comando.
  - Tipo del comando:
    - ‘const’ (valor constante, viene definido en el ‘valor del comando’).
    - ‘var’ (valor variable, se obtiene en tiempo de ejecución).
  - Valor del comando: campo a tener en cuenta en caso de que el tipo de comando sea ‘const’ (constante).

El diseño se ha llevado a cabo pensando en un futuro soporte para varias placas Arduino a la vez, aunque actualmente solo se puede soportar una durante la ejecución de un juego. En el caso de tener varias placas, el campo ‘identificador’ de ‘Arduino’ marcará a qué placa Arduino se hace referencia en cada momento. El campo ‘puerto COM’ de ‘Arduino’ indica por su parte el puerto COM del ordenador al que la placa Arduino está conectada, es decir, el puerto con el que hay que realizar la conexión.

Todos los datos de los juguetes y sus dispositivos electrónicos los define el desarrollador del juego o usuario mediante el asistente gráfico [TOYVision web]. El ‘identificador del fiducial’ asociado, el ‘identificador de la placa Arduino’ y el ‘terminal’ relativo a los dispositivos electrónicos se asignan directamente por el asistente gráfico. Por el contrario, el ‘nombre del juguete’ y el resto de los datos no nombrados relativos a los dispositivos electrónicos, incluyendo los comandos, los define el usuario.

3.2 Implementación del Gestor de Juguetes

Toda la información definida en el apartado anterior se condensa en el fichero de configuración *toys.xml* generado por el asistente gráfico (para conocer la estructura de este fichero véase el Anexo A, sección A.2). El asistente gráfico se encarga de asignar automáticamente un identificador de fiducial, una placa Arduino y un terminal de la placa únicos para cada conjunto juguete-dispositivo electrónico. Si el juguete no tiene ningún dispositivo electrónico embebido, esa parte de la información no aparece.

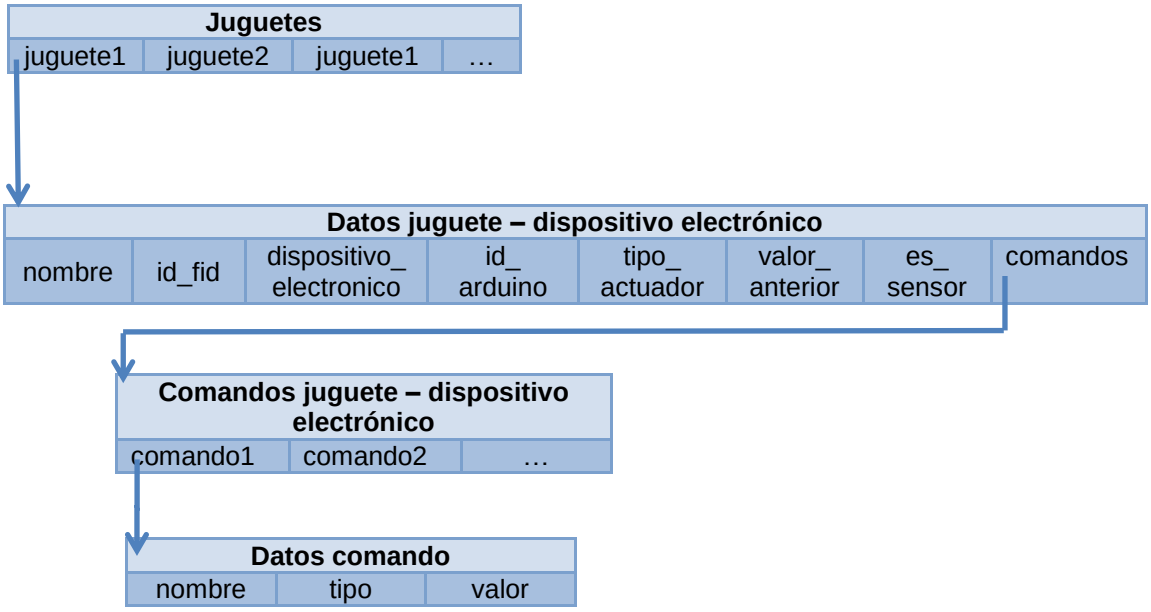
Para almacenar los datos contenidos en el fichero de configuración *toys.xml* se crean los siguientes vectores como estructura de datos:

- Un vector que guarde el identificador de cada placa Arduino junto con el puerto COM al que está conectada:

Puertos Arduinos			
Puerto_Arduino0	Puerto_Arduino1	…	Puerto_Ardduino4

Cada celda indica el puerto COM al que está conectada una placa Arduino, coincidiendo el identificador de dicha placa Arduino con el índice del vector donde se encuentre la celda. Esta estructura de datos permitirá en el futuro soportar hasta cinco placas Arduino simultáneamente.

- Un conjunto de vectores que contengan toda la información relativa a los juguetes descrita en el apartado anterior:



  - **Juguetes:** contiene la referencia a toda la información de cada juguete definido. Como se puede observar en el ejemplo, 'juguete1' aparece dos veces. Esto se debe a que se reserva un contenedor por cada combinación juguete-dispositivo electrónico. Por tanto, si un juguete lleva asociados dos dispositivos electrónicos, ocupa dos posiciones en el vector.

  - **Datos juguete – dispositivo electrónico:** relaciona los datos de un juguete con un dispositivo electrónico embebido en él (en el caso de que lo tenga):
    - nombre: nombre del juguete.
    - id_fid: identificador del fiducial adherido a la base del juguete.
    - dispositivo_electronico: nombre del dispositivo electrónico.

- id_arduino: identificador de la placa Arduino a cuyo terminal está conectado el dispositivo electrónico.
- tipo_actuador: tipo del dispositivo electrónico.
- valor_anterior: valor necesario en el caso de los sensores, tema desarrollado en el Capítulo 5 de esta memoria.
- es_sensor: indica si el dispositivo es un sensor (1) o un actuador (0).

El terminal de la placa Arduino al que el dispositivo electrónico de un juguete está conectado lo indica el índice del vector 'juguetes' en el que se encuentren los datos del juguete para un determinado dispositivo electrónico.

- **Comandos juguete – dispositivo electrónico:** contiene los comandos definidos por el usuario para un dispositivo electrónico embebido en un juguete.

- **Datos comando**: cada comando define un estado distinto para el dispositivo electrónico al que está asociado. Cuando se dé una orden a un actuador o se reciban datos de un sensor, se debe precisar el comando al que se esté haciendo referencia, para saber si hay que aplicar el dato dado en ejecución (el tipo del comando es 'var') o el definido previamente en el fichero *toys.xml* (el tipo del comando es 'const').

Existen dos vectores 'juguetes': uno para los dispositivos electrónicos digitales y otro para los dispositivos electrónicos analógicos. La razón de que existan dos vectores del mismo tipo es la siguiente.

Tal y como se ha explicado anteriormente, el índice del vector 'juguetes' indica el terminal al que está conectado el dispositivo electrónico embebido en un juguete. Si se tuviera un solo vector 'juguetes', dado que en la placa Arduino el número de ciertos terminales digitales y analógicos coincide, también podría hacerlo la posición a ocupar por dos juguetes dentro del vector que nos ocupa (tal sería el caso de un juguete con un dispositivo electrónico digital y otro con uno analógico, ambos con el mismo número como terminal). Es decir, solo se podría almacenar los datos de uno de los dos juguetes.

Si se decidiera crear un solo vector con toda la información, el índice ya no podría indicar el terminal al que está conectado un dispositivo electrónico, y el tiempo de acceso al dato en cuestión sería lineal (O(n)) al tener que recorrer todo el vector. Del modo en el que está implementado sin embargo, el tiempo de acceso pasa a ser constante (O(1)) ya que el acceso es directo. Esta decisión cobra gran importancia ya que TOYVision necesita un alto rendimiento computacional para funcionar a una velocidad satisfactoria.

Para la definición de esta estructura de datos se ha creado la clase *ToysXML*. Los métodos contenidos en esta se encargan de inicializar los vectores, leer el fichero *toys.xml* utilizando la librería *TinyXML* [TinyXML tutorial] y guardar esos datos de forma apropiada en los vectores presentados. Una vez contenida toda la información en los vectores y gracias a la estructura diseñada, se puede acceder a cualquier dato

relativo a los juguetes y sus dispositivos electrónicos desde cualquier parte del código de forma directa.

Además, existe un último método que muestra la información de los vectores definidos, de forma que al ejecutar el *framework* TOYVision se sabe qué datos se van a utilizar durante la ejecución del mismo.

Dado el tamaño del *framework* TOYVision, crear un nuevo método o pasar un nuevo atributo como parámetro se convierte en un gran problema, ya que conlleva cambiar toda línea de código que, ya sea directa o indirectamente, necesite hacer referencia a ese nuevo método o atributo.

Por esta razón, y dado que se accede a los métodos de la clase *ToysXML* desde otras dos clases distintas (ver el Anexo A para el diagrama de clases) y desde el *main()*, la clase *ToysXML* es de tipo *singleton*. La ventaja de este tipo de clases es que se hace una única instancia, de manera que cuando se llama al constructor por primera vez, se declaran e inicializan las variables en función de las inicializaciones implementadas en dicho constructor. A partir de la segunda llamada al constructor sin embargo, se sigue trabajando sobre las mismas variables antes inicializadas y modificadas, es decir, las mismas variables se pueden leer y modificar desde cualquier parte del código sin necesidad de pasarlas como parámetros.

Capítulo 4. Gestor de XMLSocket

Este cuarto capítulo describe primeramente el análisis del nuevo protocolo de comunicación creado para conseguir la bidireccionalidad y los nuevos mensajes XML buscados, explicando su implementación en un segundo apartado.

El análisis completo del Gestor de XMLSocket se muestra de forma detallada en el Anexo A (sección A.1) de este documento, donde además se puede ver su relación con el resto de las partes del proyecto.

4.1 Diseño del Gestor de XMLSocket

Uno de los objetivos fundamentales de este proyecto consiste en lograr la comunicación bidireccional y basarla en el intercambio de mensajes XML.

Tal y como se ha comentado en el Capítulo 2, el *framework* TOYVision mantenía dos canales de comunicación distintos para poder enviar dos tipos distintos de mensajes (un *socket* UDP para mensajes TUIO y un *socket* TCP para mensajes XML), lo que conllevaba varios inconvenientes:

- El uso de dos canales de comunicación requería mayor potencia computacional y mayor dificultad a la hora de programar los juegos en la API: había que estar pendiente de lo recibido por los dos canales de comunicación y de leer los dos tipos distintos de mensajes.

- La comunicación unidireccional no permitía que la API pudiera enviar mensajes al *framework* TOYVision, por lo tanto solo era posible utilizar juguetes pasivos.

- El protocolo de transporte UDP no tiene fiabilidad. Dado que no está orientado a conexión, no existen mecanismos construidos dentro del protocolo para detectar (y corregir) problemas tales como mensajes perdidos o paquetes de mensajes desordenados.

Por lo tanto, se ha creado un nuevo protocolo de comunicación basado en un *socket* TCP bidireccional encargado de enviar y recibir mensajes XML, que sustituye a los dos protocolos anteriores. De este modo se superan todas las limitaciones anteriores:

- Al estar basado en el envío de mensajes XML, ya no son necesarios dos tipos de mensajes ya que se puede añadir toda la información que se desee dentro de un mismo mensaje.

Además, el uso de un único canal de comunicación reduce la potencia computacional necesaria y facilita la creación de juegos en la API: solo hay un tipo de mensajes que leer y un canal de comunicación al que atender.

- La comunicación pasa a ser bidireccional. Por lo tanto, ya es posible tanto enviar mensajes desde el *framework* TOYVision a la API como recibirlos desde esta. Esto permite que el *framework* TOYVision conozca los factores necesarios para poder actuar sobre el juego a través de la placa Arduino.

- Por último, el protocolo de transporte es TCP, es decir, orientado a conexión, por lo que existen mecanismos para detectar y corregir problemas como mensajes perdidos o paquetes desordenados. De este modo, se consigue la fiabilidad anteriormente carente.

Los nuevos tipos de **mensajes XML** contienen la información de los eventos relacionados con añadir, modificar y eliminar los siguientes objetos:

- Fiduciales
  - Named Token (Figura 8. a).
  - Simple Token (Figura 8. b).
  - Constraint Token (Figura 8. c).

- *Fingers* (Figura 9.a).

- *Blobs* (Figura 9.b).

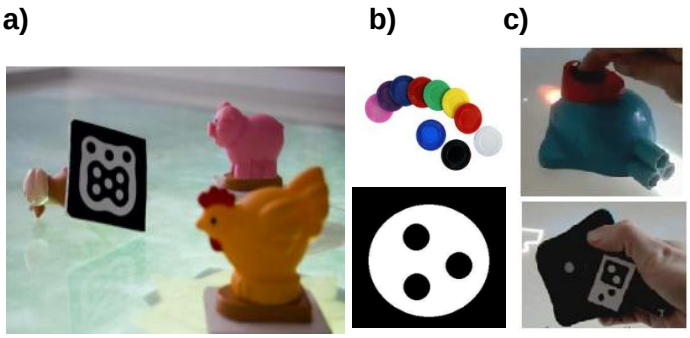


Figura 8. a) Named token: juguete con un fiducial en su base. b) Simple token: ficha con un fiducial simplificado en su base. c) Constraint token: juguete con un fiducial en su base al que se le puede quitar/poner o modificar el estado de una de sus piezas.

Figura 9. a) fingers formados con los dedos de una mano. b) blobs hechos con recortes de cartón

La información enviada para cada objeto dentro de cada uno de los eventos citados es:

- Fiduciales: nombre del juguete, número de sesión, posición en el eje x, posición en el eje y, y orientación.
- *Fingers*: número de sesión, posición en el eje x y posición en el eje y.
- *Blobs*: número de sesión, posición en el eje x, posición en el eje y, orientación y área (la forma de los *blobs* se envía informando de todo el contenido en la pantalla en un mapa de bits mediante un mensaje aparte, también en lenguaje XML).

Como se puede observar, en el caso de los fiduciales ya no se envía su identificador, sino su nombre (asignado por el usuario anteriormente en el asistente gráfico). De esta manera, cuando se programa un juego en la API es mucho más sencillo saber a qué juguete se está haciendo referencia.

La estructura de estos mensajes se puede ver en detalle en el Anexo A, sección A.3.

4.2 Implementación del Gestor de XMLSocket

Los **mensajes** en formato **XML** se crean en las clases desde las que se formaban los mensajes TUIO (*FiducialFinder* y *FidtrackFinder*). Estos mensajes se forman como una cadena de texto.

En la **conexión bidireccional**, el *framework* TOYVision actúa como servidor y la API lo hace como cliente. Para hacer la conexión se crea un *thread* (hilo de ejecución) que se ocupe de la parte de la clase *XMLSocket* en la que se crea y cierra la conexión, y se reciben mensajes. El envío de mensajes, también implementado en la clase *XMLSocket*, se hace desde el hilo de ejecución central. Cuando se cierra la conexión, se vuelve a crear una nueva a la espera de que la API la acepte, así, se evita que se tenga que cerrar todo el programa para volver a ejecutarlo si ocurre algún problema en la mesa. Para hacer todo lo relacionado con el *socket* (conexión, cierre de conexión, envío y recepción de mensajes) se utiliza la librería *winsock2.h*, y para lo relacionado con el *thread* se utiliza la librería *process.h*.

Al igual que ocurre con la clase *ToysXML*, la clase *XMLSocket* es de tipo *singleton*. Las razones vuelven a ser evitar tener que modificar todo el código del *framework* TOYVision. Además, al poder tener únicamente una instancia de la clase, se asegura tener un único *socket* en ejecución.

Aunque el nuevo protocolo de comunicación sea claramente mejor, no se ha eliminado el protocolo TUIO para no perder la compatibilidad con los juegos creados previamente. De manera que escribiendo la línea adecuada en el fichero de configuración *reactivision.xml* (generado por el asistente gráfico), se puede elegir entre utilizar un protocolo de comunicación u otro. Las líneas a utilizar son las siguientes:

- Para utilizar el protocolo de comunicación TUIO:
  <com_protocol communication="TUIO" />

- Para utilizar el protocolo de comunicación XMLSocket:
  <com_protocol communication="XMLSocket" />

Ha de tenerse en cuenta que si se escoge el protocolo de comunicación TUIO, todas las variaciones hechas en este proyecto no se reflejarán en ningún aspecto, teniendo solo las funcionalidades del *framework* ReacTIVision original.

Capítulo 5. Gestor de Arduino

En este capítulo se presenta en un primer apartado el diseño realizado para dar solución a la problemática de gestionar diferentes sensores y actuadores embebidos en los juguetes, utilizando para ello una placa Arduino. En el segundo apartado se describe la programación de la placa Arduino y la comunicación entre el *framework* TOYVision y la placa.

El análisis de este Gestor de Arduino se muestra de forma detallada en el Anexo A (sección A.1) de este documento, donde además se puede ver su relación con el resto de las partes del proyecto.

5.1 Diseño del gestor de Arduino

Para poder utilizar y controlar juguetes activos dentro de TOYVision, se han de embeber dispositivos electrónicos en ellos, por lo que es necesario el uso de un conversor analógico-digital. A continuación en primer lugar, se detallan las razones por las que se ha escogido la placa Arduino UNO como conversor analógico digital junto con el diseño de su programación, y, en segundo lugar se explica el diseño realizado para hacer la conexión de dicha placa con el *framework* TOYVision.

5.1.1 Placa Arduino como conversor analógico-digital

Aunque en el mercado hay numerosos conversores analógico-digitales como Amicus [Amicus web] o TinyCLR [TinyCLR web], en este proyecto se ha escogido la placa Arduino [Arduino web] por las razones que se presentan a continuación.

Arduino es una plataforma de *hardware* libre, basada en una placa con un micro-controlador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

Dado que es *hardware* libre, el precio de una placa no es elevado y el entorno de programación es gratuito, razones por las que Arduino actualmente está recibiendo una gran aceptación. La comunidad formada alrededor de Arduino ayuda a la comprensión y a la resolución de problemas, por lo que su uso se hace más sencillo. Todos estos factores hacen de la placa Arduino una muy buena opción para trabajar con el *framework* TOYVision.

Sin embargo, hay que mencionar que las placas Arduino también presentan ciertas desventajas. El inconveniente más importante a destacar es la diferencia de velocidad que hay entre la placa Arduino y el canal de comunicación (la primera va

mucho más rápido que el segundo). Este hecho ha sido necesario tenerlo en cuenta durante la adición de la placa al proyecto.

Arduino tiene varios tipos de placas. En este proyecto se decidió escoger la placa Arduino UNO ya que es la placa más sencilla y barata (~25 €), pero que posee las funciones necesarias para cumplir los objetivos fijados.

Arduino UNO es capaz de controlar actuadores y sensores tanto digitales como analógicos. Sin embargo, en este proyecto se han escogido utilizar actuadores únicamente digitales y sensores digitales y analógicos. La razón radica en que los actuadores analógicos no eran necesarios y además, presentaban ciertas dificultades a la hora de hacer el reparto de terminales (explicado a continuación) ya que no todos los terminales permiten ser utilizados con actuadores analógicos.

En el caso de TOYVision, la placa Arduino UNO debe tener un programa cargado en memoria en el que:

- Es necesario hacer la configuración de los terminales definiendo a qué tipo de actuadores y sensores van dirigidos cada uno de ellos, ya que se usan distintos tipos de instrucciones para encender un *led*, activar un *servo* o leer el estado de un interruptor, por ejemplo.

- Se debe gestionar la recepción de mensajes para modificar el estado de los actuadores, y el envío de mensajes para dar a conocer los datos captados por los sensores. Estos mensajes están compuestos por cinco caracteres en el caso de los actuadores y por seis en el caso de los sensores. Los dos primeros caracteres indican el terminal al que se hace referencia y los tres siguientes indican el valor leído (de sensores) o a aplicar (sobre actuadores). En el caso de los sensores, el sexto carácter es de terminación.

Aunque el fin de cada terminal esté definido, la placa no puede conocer *a priori* qué terminales van a estar ocupados y cuáles van a estar libres. En el caso de los terminales dirigidos a actuadores esto no conlleva ningún problema, ya que si no se aplican valores sobre ellos, no ocurrirá nada.

Sin embargo, los terminales dirigidos a sensores, aun cuando no tienen nada conectado, siempre captan algún valor, lo que lleva a tener valores no reales. Para dar solución a esta problemática, se asocia un *booleano* a cada uno de estos terminales para saber si dicho terminal está realmente ocupado por un sensor o no, es decir, se indica si el valor captado es válido. Inicialmente, todos estos *booleanos* tienen valor cero, de manera que es el *framework* TOYVision el encargado de ponerlos a uno en el caso de que el terminal correspondiente vaya a tener un sensor conectado.

En el anexo B (sección B.1) se muestra detalladamente la placa Arduino UNO junto con el reparto de terminales hecho para este proyecto.

5.1.2 Conexión con la placa Arduino

Al elegir la placa Arduino como conversor analógico-digital aparecían varios aspectos a tener en cuenta:

- Se debe poder lidiar con un número y tipo desconocidos de sensores y actuadores, ya que dependiendo del juego que quiera hacer el desarrollador, utilizará unos u otros.

- Se debe poder hacer una conexión bidireccional con la placa Arduino, para poder tanto enviar órdenes a los actuadores como recibir información de los sensores.

- Dado que las órdenes dirigidas a los actuadores deben llegar desde la API hasta la placa Arduino y los datos captados por los sensores de la placa Arduino deben llegar hasta la API, el *framework* TOYVision debe actuar como intermediario traduciendo los mensajes enviados por uno y otro.

Para dar solución al segundo punto es necesario un nuevo protocolo de comunicación bidireccional. Dado que la placa Arduino se conecta al ordenador mediante un puerto COM, el protocolo está basado en la comunicación por puerto serie y en mensajes de tipo texto, ya que la información a intercambiar con la placa es de bajo nivel.

Para dar solución al primer y tercer puntos presentados, es necesaria una gestión de datos, de la que se encarga este Gestor de Arduino, en la que el Gestor de Juguetes cobra una gran importancia. Dado que toda la información relativa a los dispositivos electrónicos a utilizar está guardada en la estructura de datos del Gestor de Juguetes explicada en el Capítulo 3 de la memoria, se pueden conocer los dispositivos electrónicos que se van a utilizar durante la ejecución del juego y de qué forma.

Al disponer de esta información, el Gestor de Arduino ya es capaz de “activar” aquellos terminales dirigidos a sensores que vayan a ser utilizados mediante una cadena de texto.

Además, cuando el Gestor de XMLSocket recibe una orden en formato XML (proveniente de la API) dirigida a un actuador, reenvía la orden al Gestor de Arduino, quien se encarga traducir dicho mensaje a una cadena de texto que la placa sepa interpretar.

De forma inversa, cuando el Gestor de Arduino recibe una cadena de texto desde la placa con la información que un sensor haya captado, traduce esa cadena de texto en un mensaje XML para poder enviarlo a través del Gestor de XMLSocket a la API.

5.2 Implementación del gestor Arduino

A continuación se explica cómo se ha programado la placa Arduino UNO y la implementación hecha dentro del *framework* TOYVision para permitir la comunicación de la placa con TOYVision.

5.2.1 Programación de la placa Arduino UNO

La placa Arduino UNO tiene un lenguaje de programación propio basado en C/C++ [Lenguaje Arduino web] y un entorno de programación gratuito que se puede descargar desde la Web [Arduino 1.0.1 web]. Tiene una memoria de almacenamiento de 1KB, una RAM de 2KB y una velocidad de procesamiento de 16MHz, lo que implica que existe la posibilidad de que la placa procese los datos más rápido de lo que viajan a través del puerto serial (depende de la frecuencia de lectura de datos del ordenador al que la placa esté conectada).

Tal y como se ha comentado en el apartado anterior, al programar la placa Arduino es necesario primeramente hacer el reparto de terminales, para después pasar a la gestión de mensajes. En el caso del programa hecho en este proyecto para la placa Arduino, esta gestión de mensajes se basa en un bucle infinito donde:

- si se desea modificar el estado de un **actuador digital**, se le envía a la placa una cadena de texto de cinco caracteres indicando (en los dos primeros caracteres) sobre qué terminal actuar y (en los tres últimos caracteres) qué valor aplicar: encender/apagar un *led*, girar un número de grados un *servo*… Al recibir la cadena, dependiendo del tipo de actuador al que la orden vaya dirigida, se da la orden con una instrucción u otra, indicando el terminal sobre el que actuar y el valor a aplicar.

- para leer un **sensor digital**, la placa está programada para que devuelva el valor de un sensor cuando este haya cambiado. Es decir, si un interruptor está pulsado, la placa Arduino no informará de un nuevo estado hasta que el interruptor deje de estarlo. La información se envía mediante una cadena de texto de seis caracteres, indicando (en los dos primeros caracteres) el terminal al que está conectado el sensor y (en los tres siguientes caracteres) el valor captado por el mismo, añadiendo el carácter '/' como terminación.

- en el caso de los **sensores analógicos** también interesa que la placa notifique cuándo el valor de lo captado por un sensor ha cambiado, sin embargo, los valores captados por los sensores analógicos varían mucho y muy deprisa al estar fluctuando continuamente dentro del mismo rango de valores. Esto conlleva a tener demasiados datos que pueden colapsar el canal de comunicación. Por esta razón, en el caso de los sensores analógicos, la placa notifica un cambio de valor cuando este nuevo valor tiene una diferencia de, como mínimo, 75, con respecto al último notificado. Tal y como ocurre con los sensores digitales, se notifica el terminal y el

valor captado mediante una cadena de texto de seis caracteres, con el carácter '/' como terminación.

Para activar los terminales dirigidos a sensores tal y como se ha descrito en el apartado anterior, la placa ha de recibir previamente una cadena de texto de cinco caracteres de los cuales los dos primeros deben ser '20' y los tres últimos indicar el terminal a activar:

- Si el sensor a activar es digital, esos tres últimos caracteres serán: 00X,
- Si el sensor a activar es analógico, esos tres últimos caracteres serán: 01X,

siendo X el terminal a activar.

Los actuadores y sensores probados con el programa Arduino realizado en este proyecto se especifican en el Anexo B, sección B.2. En la sección B.3 además, se indican los circuitos a montar para cada uno de esos dispositivos electrónicos.

5.2.2 Comunicación entre la placa Arduino y el *framework* TOYVision

La comunicación entre la placa Arduino y el *framework* TOYVision se centra en dos grandes partes:

- la **conexión**, que se realiza mediante la clase *Serial* del *framework* TOYVision, descargada desde la web oficial de Arduino [Arduino Serial web]. Sus métodos se encargan de hacer la conexión con el puerto COM, enviar mensajes a dicho puerto y recibir notificaciones desde el mismo.

- la **gestión de los datos** a intercambiar entre ambos, que se consigue gracias a la clase *GestorArduino* del *framework* TOYVision encargada de:
    - preparar la conexión con la placa Arduino.
    - formar las órdenes precisas a enviar a la paca Arduino en función de los mensajes XML recibidos desde el Gestor de XMLSocket (que a su vez recibe las órdenes desde la API).
    - tratar los datos captados por los sensores para después transmitirlos, mediante el Gestor de XMLSocket, a la API por mensajes XML.

Para todo ello se hace uso de la información previamente recogida en el Gestor de Juguetes (Capítulo 3) leída del fichero de configuración *toys.xml* (Anexo A, sección A.2).

A continuación se describe con más detalle la clase *GestorArduino,* en la que se encuentran:

- un **constructor** que se encarga de:

    1. indicar el **puerto COM** en el que la placa Arduino se encuentra conectada para así hacer la conexión desde la clase *Serial*.

2. comprobar cuáles son los **terminales dirigidos a sensores** (tanto analógicos como digitales) que se van a utilizar, para así poder activarlos mediante el envío de mensajes, tal y como se ha explicado en el subapartado anterior.

3. crear un *thread* (hilo de ejecución) encargado de hacer una escucha permanente para poder captar todos los mensajes enviados desde la placa.

- el control hecho sobre los **actuadores**, donde, tras haber recibido un mensaje XML proveniente del Gestor de XMLSocket indicando qué acciones llevar a cabo,

    - analiza el **mensaje XML** con la ayuda de la biblioteca *TinyXML* [TinyXML tutorial] (la estructura de dicho mensaje XML se puede ver en el anexo A, sección A.3) para así obtener:
        - el **nombre del juguete**.
        - el **actuador** correspondiente sobre el que actuar.
        - el **comando** del actuador al que se hace referencia.
        - el **valor** a aplicar. Este dato viene en el mismo mensaje XML en el caso de que el tipo del comando en cuestión sea variable (‘var’). Si el tipo del comando está definido como constante (‘const’), se tomará el valor ya definido en el fichero de configuración *toys.xml* y contenido en la estructura de datos del Gestor de Juguetes.

    - busca el **terminal** en el que se encuentra el juguete y el actuador referidos (se recuerda que el terminal coincide con el índice del vector donde se encuentra la información relativa a los juguetes).

    - envía un **mensaje** a la placa con el terminal sobre el que actuar y el valor a aplicar (con la cadena de texto de cinco caracteres citada en el subapartado anterior).

- la lectura de los **sensores analógicos y digitales** y el reenvío de la información obtenida, para lo cual:

    - el *thread* creado en el constructor se queda **escuchando** continuamente hasta que recibe un mensaje (de seis caracteres) proveniente de la placa. Este mensaje llega carácter por carácter, por lo que el Gestor de Arduino seguirá leyendo hasta recibir el carácter de terminación ‘/’.

    - se **analiza** el mensaje (descrito en el subapartado anterior) proveniente de la placa y se obtiene:
        - el **terminal** al que está conectado el sensor.
        - el **valor** captado por el sensor.

    - se **certifica** que el valor obtenido por el sensor sea válido (distinto al valor recibido por el mismo sensor en la anterior notificación). Aunque tal y como se ha dicho anteriormente, la placa está

programada para en el caso de los sensores solo notifique cambios de valores, dado que la comunicación en ocasiones falla, se puede llegar a recibir la misma información dos veces seguidas. Esta certificación se realiza utilizando el campo **'valor_anterior'** definido en la estructura de datos de la clase *ToysXML* (Capítulo 3, sección 3.2).

- se **prepara** el **mensaje XML** a enviar al Gestor de XMLSocket para, a través de este, poder informar a la API de la nueva variación (la estructura de dicho mensaje XML se puede ver en el anexo A, sección A.3). Los datos de este mensaje (juguete y sensor implicados) se obtienen de la estructura de datos del Gestor de Juguetes, utilizando el terminal como índice de dicha estructura. Una vez se tienen estos datos, se envía un mensaje XML a Gestor de XMLSocket con la información de:
  - el **nombre del juguete**.
  - el **sensor** del que proviene el nuevo valor (recordemos que un juguete puede tener más de un actuador y/o sensor embebidos).
  - el **valor** obtenido (si el comando del sensor está definido como 'var') o el **estado** al que hace referencia el valor obtenido (si el comando del sensor está definido como 'const'; si no se acerca a ningún valor predefinido se devuelve 'null').

Capítulo 6. Resultados obtenidos

En este capítulo se presentan los resultados obtenidos en este proyecto mediante un juego inspirado en los juegos de tablero de *Dragones y Mazmorras*. En el primer apartado se explica en qué consiste el juego, mientras que en el segundo se describe su funcionamiento.

6.1 Descripción del juego

Como resultado de lo realizado en este Proyecto Fin de Carrera se ha podido realizar un juego para NIKVision que aprovecha las nuevas funcionalidades implementadas en el *framework* TOYVision [Demo juego web].

El escenario del juego está formado por un caballero (Figura 10.1), un rey, un cofre (Figura 10.4) que guarda una espada mágica, una columna y un dragón (Figura 10.3).

Tal y como se muestra en la Figura 10, al comenzar el juego, se posicionan los objetos mencionados en distintas partes de la mesa, tras lo cual el dragón comienza a girar sobre sí mismo poco a poco y dos dados (Figura 10.2) aparecen en la mesa. Dichos dados muestran un número aleatorio que indica el número de casillas que puede recorrer el caballero o el rey.

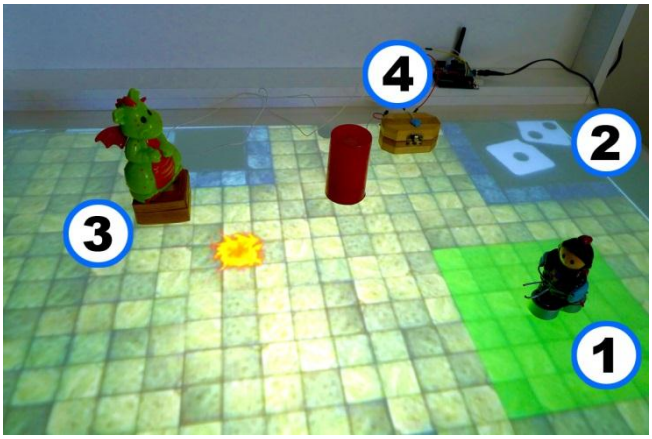


Figura 10. Escenario inicial del juego que emula a Dragones y Mazmorras: 1- caballero; 2- dados que indican el número de casillas a mover; 3- dragón; 4- cofre con una espada en su interior

El objetivo del juego es matar al dragón, para lo cual se ha de llegar primeramente hasta el cofre para conseguir la espada y, finalmente, llegar hasta el dragón. La dificultad radica en que si el dragón mira directamente a un jugador, le lanza una bola de fuego (tal y como aparece en la Figura 10), con lo que el jugador

muere. Esto se evita si dicho jugador se encuentra oculto por la columna (roja en la Figura 10), caso en el que la bola de fuego no le alcanza.

6.2 Funcionamiento del juego

Tal y como se ve en la Figura 11, el caballero y el rey llevan un fiducial en su base y junto a este, un *led* infrarrojo. El *led* va conectado a un interruptor magnético (se activa acercando un imán) colocado en la parte frontal del juguete. Dado que el *led* puede estar encendido o apagado, estos juguetes pueden modificar su estado, lo que les define como *constraint token*.

Figura 11. Caballero con un fiducial y un led infrarrojo en su base

El dragón, la columna y el cofre llevan adherido a sus respectivas bases un fiducial, por lo que son de tipo *named token*. El dragón y el cofre además, son juguetes activos ya que llevan un *servo* para que se puedan mover de forma independiente. En la Figura 12 se muestra cómo el *servo* del cofre va enganchado a la tapa del mismo, para que cuando el *servo* gire, la tapa se abra (la espada que se encuentra en su interior lleva además un imán). En el caso del dragón, el *servo* se encuentra bajo el juguete para que el dragón pueda girar sobre sí mismo.

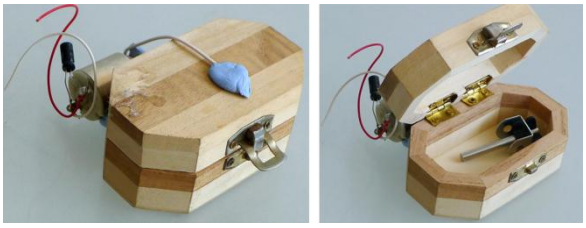


Figura 12. Cofre con un servo en su parte trasera y una espada en su interior

En la Figura 13 se presenta el fichero *toys.xml* utilizado para la ejecución del juego, cuya información se puede resumir de la siguiente manera (se muestran los datos que solo interesan al *framework* TOYVision):

- Arduino con identificador ‘1’ conectado al puerto ‘COM6’.

- Named tokens
    - Cofre:
        - Nombre: ‘chest’.
        - Identificador: ‘2’.

        - Actuador: nombrado como ‘cover’, es un *servo* ‘non_continuous’ (no continuo) conectado al terminal ‘10’ de la placa Arduino con identificador ‘1’. Define dos comandos:
            - ‘close’ de tipo ‘const’ con valor ‘0’.
            - ‘open’ de tipo ‘const’ con valor ‘180’.
    - Columna:
        - Nombre: ‘column’.
        - Identificador: ‘3’.
    - Dragón:
        - Nombre: ‘dragon’.
        - Identificador: ‘4’.
        - Actuador: nombrado como ‘motor’, es un *servo* ‘non_continuous’ (no continuo) conectado al terminal ‘11’ de la placa Arduino con identificador ‘1’. Define dos comandos:
            - ‘turn’ de tipo ‘var’.
            - ‘reset’ de tipo ‘const’ con valor ‘0’.

- Constraint tokens
    - Caballero:
        - Nombre: ‘knigth’.
        - Identificador: ‘0’.
    - Rey:
        - Nombre: ‘king’.
        - Identificador: ‘1’.

Con esta información el *framework* TOYVision ya es capaz de intercambiar mensajes con la API y con la placa Arduino. Al ejecutarse el *framework*, el Gestor de Juguetes lee el fichero *toys.xml* rellenando la estructura de datos de los juguetes (Capítulo 3). Cuando la API acepta la conexión del *socket* (Capítulo 4), se lee la información relativa a las placas Arduino y se establece la conexión a través del puerto COM indicado en el *toys.xml* (Capítulo 5), en este caso con el puerto ‘COM6’.

Conforme se van colocando los juguetes sobre la mesa, se envían desde el *framework* a la API los mensajes relativos a la detección de un “nuevo objeto”:

Cuando el *framework* detecta el fiducial con identificador 2, el Gestor de XMLSocket consulta al Gestor de Juguetes para averiguar el juguete que tiene el identificador 2, el cofre (chest), y así poder generar y enviar a la API este mensaje:

<fiducial_added id_session=”1 “ name=”chest “ pos_x=”x “ pos_y=”y “ orientation=”o “ />

De igual forma:

Cuando el *framework* detecta el fiducial con identificador 3:

<fiducial_added id_session=”2 “ name=”column “ pos_x=” x“ pos_y=”y “ orientation=”o“ />

Cuando el *framework* detecta el fiducial con identificador 4:

<fiducial_added id_session=”3 “ name=”dragon “ pos_x=”x “ pos_y=” y“ orientation=”o “ />

Cuando el *framework* detecta el fiducial con identificador 0:

<fiducial_added id_session=”4“ name=”knigth “ pos_x=”x “ pos_y=”y “ orientation=”o“ />

Cuando el *framework* detecta el fiducial con identificador 1:

<fiducial_added id_session=”5 “ name=”king“ pos_x=” x“ pos_y=”y “ orientation=”o “ />

Los campos ‘pos_x’, ‘pos_y’ y ‘orientación’ contienen la posición y orientación de cada juguete captados por la cámara.

Tras el envío de estos mensajes, la API ya conoce la ubicación de cada objeto y comienza el juego: aparecen los dados indicando el número de casillas que pueden mover el caballero y el rey, y el dragón comienza a girar sobre sí mismo.

Cuando el caballero y el rey son movidos por el jugador, se envía un mensaje desde el *framework* a la API de “objeto modificado”:

Cuando el *framework* detecta que el fiducial movido tiene identificador 0, el Gestor de XMLSocket, con la información del Gestor de Juguetes, genera este mensaje:

<fiducial_updated id_session=”6 “ name=”knigth “ pos_x=”x “ pos_y=”y “ orientation=”o“ />

Cuando el *framework* detecta que el fiducial movido tiene identificador 1, el mensaje generado es el siguiente:

<fiducial_updated id_session=”7 “ name=”king“ pos_x=” x“ pos_y=”y “ orientation=”o “ />

Nuevamente, en los campos ‘pos_x’, ‘pos_y’ y ‘orientacion’ aparece la posición relativa a los juguetes.

Por otra parte, para que el dragón gire continuamente, la API manda cada cierto tiempo el siguiente mensaje al Gestor de XMLSocket a través del *socket*:

<toy=”dragon” actuator=”motor” status=”turn” value=”10”/>

donde se indica que se debe aplicar al actuador de nombre ‘motor’ asociado al juguete ‘dragon’ el valor ‘10’, es decir, se debe girar el servo hasta la posición de 10º. El Gestor de Arduino analiza este mensaje con la ayuda de la estructura del Gestor de Juguetes, obteniendo los siguientes datos:

- Terminal sobre el que actuar: ‘11’
- Valor a aplicar: ‘10’ (ya que el tipo del comando es ‘var’ se coge el valor contenido en ‘value’)
- Puerto en el que la placa está conectada según su identificador (1 en este caso): ‘COM6’

De este modo la orden a enviar a la placa Arduino conectada en el puerto ‘COM6’ es “11010” (la orden debe estar compuesta por cinco caracteres).

Cuando la placa Arduino recibe este mensaje lo desglosa obteniendo el terminal (‘11’) y el valor a aplicar (‘10’). Una vez obtiene el terminal puede analizar a qué tipo de dispositivo electrónico va dirigida la orden (*servos*) y aplicar en consecuencia la instrucción correspondiente.

Para que el dragón esté en continuo movimiento, la API continúa enviando este mensaje cambiando el valor de 'value' hasta llegar a 180 (valor máximo de un *servo* no continuo):

<toy="dragon" actuator="motor" status="turn" value="20"/>
<toy="dragon" actuator="motor" status="turn" value="30"/>
…
<toy="dragon" actuator="motor" status="turn" value="180"/>

Cuando llega a este último valor vuelve a poner el dragón en la posición inicial mediante el mensaje:

<toy="dragon" actuator="motor" status="reset" value=""/>

En este caso, cuando el Gestor de Arduino del *framework* analiza el mensaje, al comprobar el tipo del comando, este aparece como 'const', por lo que consulta el valor a aplicar al Gestor de Juguetes, en este caso '0'. En consecuencia, los datos obtenidos son:

- Terminal sobre el que actuar: '11'
- Valor a aplicar: '0'
- Puerto al que está conectada la placa: 'COM6'

La orden a enviar a la placa Arduino conectada al puerto 'COM6' es entonces: "11000".

Según la orientación del dragón dada inicialmente y conociendo el número de grados que este va girando, el código del juego mantiene una variable con los grados de orientación que indican hacia dónde "está mirando" el dragón en cada momento, por lo que si en alguna ocasión esa dirección coincide con la posición del caballero o del rey, le lanza una bola de fuego.

Conforme se desarrolla el juego, tanto el caballero como el rey van avanzando por las casillas, por lo que llegado el momento en el que la posición notificada por el *framework* permita a la API certificar que o bien el caballero o bien el rey están al lado del cofre (de ahora en adelante se supondrá que ha sido el caballero el que ha llegado hasta el cofre), este se debe abrir automáticamente. Para ello la API envía el siguiente mensaje:

<toy="chest" actuator="cover" status="open" value=""/>

Una vez el Gestor de Arduino recibe el mensaje, lo analiza y obtiene el terminal al que está conectado el actuador 'cover' del juguete 'chest' (10) y, tras comprobar que el tipo del comando 'open' es 'const', el valor definido en el Gestor de Juguetes, en este caso, '180'. Con esta información, envía el siguiente mensaje a la placa (conectada en 'COM6'): "10180".

Al abrirse la tapa del cofre, ya se puede coger la espada contenida en su interior y adherirla al caballero. A causa del imán de la espada, el interruptor magnético del caballero se activa encendiendo en consecuencia el *led* infrarrojo de su base. Cuando el *led* se enciende, el *framework* lo toma por un *finger* dados su tamaño y forma, por lo que el Gestor de XMLSocket envía este mensaje a la API:

<cursor_added id_session="f8 " pos_x="x " pos_y=" y"/>

donde ‘pos_x’ y ‘pos_y’ son la posición donde haya aparecido el *finger*. Dada esta ubicación y la del caballero que porta la espada, la API interpreta ese *finger* como la parte modificable del *constraint token* del juguete del caballero, por lo que sabe que el caballero ha cogido la espada.

Cuando el caballero se aleja del cofre lo suficiente, la API envía el siguiente mensaje para cerrar el cofre:

<toy=”chest” actuator=”cover” status=”close” value=””/>

que se traduce en la orden “11000” a enviar a la placa conectada al puerto ‘COM6’, ya que el actuador “cover” del juguete “chest” se encuentra conectado en el terminal ‘11’ y el comando ‘close’, que es de tipo ‘const’, tiene asociado el valor ‘0’.

De este modo, el caballero y el rey siguen avanzando enviándose por lo tanto continuos mensajes de “objeto modificado” del *framework* a la API.

Cuando la posición del caballero (que porta la espada) llega a una posición relativa al dragón que la API interpreta como “al lado”, el dragón muere y el caballero gana.

```
<toys>

  <arduinos>
    <un_arduino id="1" port="COM6"/>
  </arduinos>

  <allNamedTokens>

    <namedToken name="chest" fidID="2" copies="1" shape="rect"
      x="0.735" y="0.565" width="0.06333333333333334" height="0.115">
      <actuator name="cover" terminal="10" arduino="1"
      type="non_continuous" icon="180-Servo">
        <command name="close" type="const" value="0"/>
        <command name="open" type="const" value="180"/>
      </actuator>
    </namedToken>

    <namedToken name="column" fidID="3" copies="1" shape="circle"
      x="0.5733333333333334" y="0.625" width="0.03833333333333333"
      height="0.03833333333333333"/>

    <namedToken name="dragon" fidID="4" copies="1" shape="rect"
    x="0.41333333333333333" y="0.58" width="0.08666666666666667"
    height="0.10333333333333333">
      <actuator name="motor" terminal="11" arduino="1"
      type="non_continuous" icon="180-Servo">
        <command name="turn" type="var"/>
        <command name="reset" type="const" value="0"/>
      </actuator>
    </namedToken>
  </allNamedTokens>

  <allSimpleTokens/>

  <allConstraintTokens>

    <constraintToken name="knight" fidID="0" copies="1" shape="rect"
    x="0.5166666666666667" y="0.33166666666666667" width="0.045"
    height="0.07666666666666666">
      <AssociativeArea name="sword" radious="0.035"
      distance="0.070046286306095" angle="0.03569911267932491"/>
    </constraintToken>

    <constraintToken name="king" fidID="1" copies="1" shape="rect"
    x="0.5166666666666667" y="0.33166666666666667" width="0.045"
    height="0.07666666666666666">
      <AssociativeArea name="n" radious="0.035"
      distance="0.070046286306095" angle="0.03569911267932491"/>
    </constraintToken>

  </allConstraintTokens>

  <allDeformableTokens/>

</toys>
```

Figura 13. Fichero toys.xml para el juego del dragón

Capítulo 7. Conclusiones y trabajo futuro

En este último capítulo se resumen la consecución de objetivos, el trabajo futuro que se puede realizar a partir de este punto y la valoración personal del trabajo llevado a cabo en este Proyecto Fin de Carrera.

7.1 Consecución de objetivos

Según los objetivos planteados en el primer capítulo de esta memoria, y tal y como se ha podido observar a lo largo de la misma, se ha conseguido llevar a cabo lo propuesto:

- Se ha cambiado el protocolo de comunicación existente en el *framework* TOYVision compuesto por dos tipos de canales de comunicación, por otro protocolo compuesto por un único canal de comunicación, mucho más fiable y sencillo de manejar a la hora de hacer juegos en la API.

- Se han sustituido los dos tipos de mensajes con los que se comunicaba toda la información a la API por un único tipo de mensajes: mensajes XML. Gracias a este tipo de mensajes, no solo se ha podido englobar toda la información a enviar, sino que también se ha facilitado el hecho de poder añadir toda la información que se quiera sin ningún tipo de restricción.

- Se ha conseguido poder utilizar juguetes activos, embebiendo la electrónica adecuada en ellos. De esta manera, ahora no solo es posible detectar los juguetes posicionados sobre la mesa, sino que también se pueden detectar cambios en el ambiente y los juguetes pueden actuar de forma independiente. Es decir, las posibilidades de hacer nuevos tipos de juegos han incrementado notablemente.

Además de haber podido alcanzar los objetivos técnicos, hay que destacar que el *tabletop* NIKVision mejorado con las nuevas funcionalidades introducidas por este proyecto en el *framework* TOYVision ha tenido una gran aceptación tanto dentro de la Universidad de Zaragoza como fuera.

En la Universidad de Zaragoza se hicieron prácticas de la asignatura de Diseño y Evaluación de Interfaces del Máster de Ingeniería Informática utilizando el *framework* TOYVision con las nuevas funcionalidades. Las prácticas consistieron en aprender a hacer un juego tangible con TOYVision, donde los participantes definieron mediante el asistente gráfico los juguetes a utilizar e implementaron la lógica del juego en AS3 gracias a la información enviada por el *framework* (Figura 14).

Figura 14. Prácticas de Máster

Los alumnos aseguraron mediante un *test* realizado tras las prácticas, haber disfrutado mucho utilizando una herramienta como esta, ya que para ellos supuso una forma muy sencilla de explorar un tipo de interacción (tangible) muy novedoso sobre el cual no tenían experiencia previa

Fuera de la Universidad de Zaragoza este proyecto dio lugar a un *workshop* de un día de duración en la conferencia Internacional sobre *Tangible, Embedded and Embodied Interaction* (TEI) celebrada en Barcelona en Febrero de 2013 [TEI 2013 web]. Esta conferencia supone la referencia internacional sobre computación ubicua y tangible y reúne cada año las propuestas más importantes en este campo.

Los participantes de nuestro *workshop* pudieron diseñar y desarrollar su propio juego, utilizando para ello el *toolkit* TOYVision [Workshop TEI 2013 web]. Gracias al *framework* TOYVision creado en este proyecto, los participantes fueron capaces de pasar del concepto en papel al prototipo funcional en TOYVision con facilidad y rapidez.

Al inicio de la jornada se mostró como ejemplo el juego presentado en el Capítulo 6, lo que motivó a los participantes para hacer uno propio. Decidieron hacer un juego entre todos, creando un ambiente de grupo divertido y relajado.

Las tareas se dividieron en varias fases. En la primera idearon el concepto del juego. Decidieron hacer de él como si de una historia se tratara, teniendo que sortear un escenario laberíntico y varios enemigos para conseguir llegar al final (Figura 15).

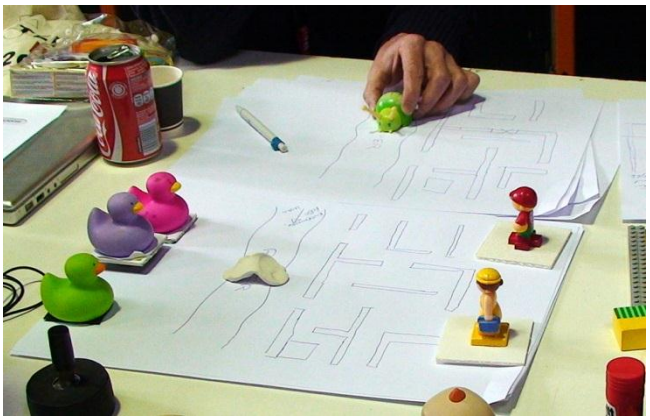


Figura 15. Fase 1 del workshop

Durante la segunda fase los participantes eligieron los juguetes que iban a utilizar para su juego y si estos serían pasivos o activos. A partir de ahí se dividieron en dos grupos para, por una parte construir los juguetes, y por la otra diseñar el escenario del juego (Figura 16).

Figura 16. Fase 2 del workshop

En la tercera fase comenzaron a definir mediante el asistente gráfico todos los juguetes que iban a utilizar, incluyendo también los dispositivos electrónicos (Figura 17).

Figura 17. Fase 3 del workshop

En la cuarta y última fase hicieron la implementación del juego, donde se encargaron de leer los mensajes provenientes del *framework* TOYVision para conocer la existencia y posición de los juguetes que hubiera sobre la mesa, y a enviar las órdenes al *framework* TOYVision para que los juguetes actuaran en el momento adecuado. Para cuando terminó la jornada, consiguieron que el juego funcionara, el cual fue presentado a todos los participantes del congreso durante la sesión de Demos (Figura 18).

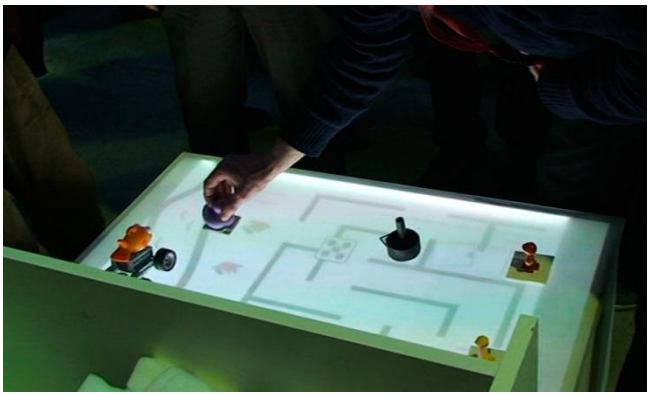


Figura 18. Fase 4 del workshop

Al finalizar, los participantes rellenaron un *test* en el que valoraron especialmente la forma en que TOYVision acercaba a las fases de creación del concepto y la implementación del prototipo.

7.2 Trabajo futuro

En esta sección se describen los posibles trabajos futuros que podrían complementar o ampliar el proyecto realizado.

- **Soporte de varias placas Arduino a la vez:** tal y como se ha dicho en el Capítulo 3 de esta memoria, el diseño de este proyecto se hizo pensando en un futuro soporte de hasta cinco placas Arduino. Por lo tanto, un posible y recomendable trabajo futuro consistiría en completar la implementación para que esto sea posible. De esta forma, se podrán tener muchos más actuadores y sensores al mismo tiempo durante la ejecución de un juego, es decir, se podrán tener juegos más complejos al ser posible tener más juguetes con un mayor número de dispositivos electrónicos embebidos. Además, sería también recomendable conectar estas placas Arduino por medio de *bluetooth*, ya que evitaría tener cables engorrosos sobre la mesa.

- **Incluir el asistente gráfico dentro del *framework* TOYVision:** actualmente, para que el desarrollador del juego defina los juguetes a utilizar y los dispositivos electrónicos que estos van a tener embebidos durante la ejecución de un juego, debe utilizar un asistente gráfico. Este asistente gráfico es un ejecutable externo, por lo que si se desea cambiar alguna característica de un solo juguete, se debe cerrar la ejecución del juego y volver a definir todos los juguetes y sus respectivos dispositivos electrónicos. De este modo, sería muy interesante poder incluir este asistente gráfico dentro del *framework* TOYVision para poder hacer cualquier modificación relativa a la definición de juguetes (y sus dispositivos electrónicos) durante la ejecución del juego, sin necesidad de parar la ejecución ni de redefinir los juguetes.

7.3 Valoración personal

Personalmente, me encuentro muy satisfecha con el trabajo realizado. He podido cumplir los objetivos inicialmente fijados y he podido trabajar en un campo de la informática que siempre me ha fascinado. La experiencia ha sido muy enriquecedora desde diversos puntos de vista:

Desde el punto de vista técnico, he podido trabajar con el *tabletop* NIKVision, una tecnología poco común con la que no es fácil llegar a trabajar. Además, he tenido que investigar cómo usar y sacar partido a una placa Arduino, lo que ha resultado muy interesante.

Gracias a que la implementación se ha centrado casi en su totalidad en tener que modificar el *framework* TOYVision, he podido experimentar lo que supone enfrentarme a un código no solo muy grande sino también modificado por varias personas, cada una con una forma distinta de programar. Además, he podido aplicar muchos de los conocimientos aprendidos a lo largo de la carrera. Dentro de estos conocimientos destaco el haber podido aplicar en un entorno real el uso de *sockets, threads*, e incluso un poco de electrónica.

En cuanto a la gestión del proyecto se refiere, he aplicado los conocimientos teóricos y prácticos adquiridos a lo largo de la carrera. He podido experimentar lo importante que resulta hacer una buena planificación y el ir modificándola conforme los

problemas aparecen. De esta forma fue posible terminar en la fecha propuesta inicialmente, algo de suma importancia dado que el *workshop* solicitado había sido aceptado.

Desde el punto de vista del resultado, no solo me siento satisfecha por haber conseguido los objetivos fijados, sino también por haber podido acudir a un congreso internacional con el trabajo realizado, pudiendo así conocer el mundo de la investigación.

Finalmente, deseo valorar toda la experiencia adquirida con mis directores, que me ha ayudado a no bloquearme al enfrentarme a un problema, mayor incluso de lo que inicialmente se planteaba, afrontándolo paso a paso. Además, gracias a ellos he aprendido cómo explicar de forma detallada un trabajo como el aquí realizado.

Anexo A. Documentación del desarrollo software

Este anexo complementa a los capítulos 3, 4 y 5 contenidos en la memoria. Explica la metodología de análisis utilizada, el fichero de configuración *toys.xml*, la configuración de mensajes y las herramientas utilizadas.

A.1 Metodología de análisis

En esta sección se explica la metodología de análisis del problema, tras la cual se definen los requisitos y los diagramas utilizados, y los pasos seguidos para desarrollar cada modelo de descripción del sistema. La metodología de análisis seleccionada es OMT [RBP96]. Según esta metodología la fase de análisis se realiza en tres pasos:

- Modelo de objetos: describe la estructura estática de los objetos del sistema (relaciones, atributos y operaciones) que se representa mediante diagramas de objetos.
- Modelo dinámico: describe los aspectos de un sistema estudiando la organización de estados y la secuencia de operaciones asociadas con los usuarios mediante diagramas de estado.
- Modelo funcional: describe las transformaciones que pueden sufrir los datos dentro del sistema, representándolo gráficamente mediante diagramas de flujo de datos.

Aunque el análisis de requisitos es propio de la metodología UML, se ha creído oportuno añadirlo a estas tres etapas de la fase de análisis, ya que permitirá estudiar más a fondo las necesidades del *framework*. Por otra parte, se ha omitido el modelo dinámico por ser prácticamente nula la interacción directa con el usuario en lo implementado en este proyecto.

A.1.1 Análisis de requisitos

La primera etapa del desarrollo consiste en establecer los requisitos que debe cumplir el proyecto para tener una mayor comprensión de los objetivos a alcanzar. Para ello, antes de comenzar con la realización del proyecto se hicieron varias reuniones con los directores para así dejar claros dichos objetivos.

A continuación, se muestran los requisitos funcionales y no funcionales que se determinaron como resultado de las reuniones mencionadas. Mientras que los requisitos funcionales reflejan las distintas partes a implementar, los requisitos no funcionales definen las restricciones tanto *software* como *hardware* del proyecto.

Requisitos funcionales:

RF1- Crear una especificación en lenguaje XML para informar de todas las manipulaciones que recoge el *framework* TOYVision: adición, modificación y eliminación de los fiduciales (normales y *simple tokens*), *fingers* y *blobs*, y uso de sensores y actuadores.

RF2- Implementar un socket TCP bidireccional que comunique el *framework* TOYVision y la API.

RF3- Implementar una conexión bidireccional para la comunicación entre el *framework* TOYVision y los sensores y actuadores embebidos en los juguetes.

RF4- Gestionar la información intercambiada entre el *framework* TOYVision y los sensores y actuadores.

RF5- Programar la placa Arduino para el control de los actuadores y sensores embebidos en los juguetes.

RF6- Gestionar la información relativa a los juguetes:

- idear un fichero de configuración que detalle la información relativa a los juguetes.
- guardar y gestionar la información relativa a los juguetes contenida en el fichero de configuración.

RF7- Filtrar eventos falsos debidos al ruido de la cámara para aumentar la robustez de la detección de las manipulaciones de los juguetes.

Requisitos no funcionales:

RNF1- Modificar el código del *framework* TOYVision existente en lenguaje C++.

RNF2- Usar el entorno de programación Microsoft Visual Studio 2005.

RNF3- Utilizar una placa Arduino UNO para el control de los actuadores y sensores.

RNF4- Utilizar el entorno de programación Arduino 1.0.1 para Windows [Arduino 1.0.1 web] para implementar y cargar el código en la placa Arduino.

RNF5- Imponer que el fichero de configuración relativo a los juguetes sea de tipo XML y almacenarlo en la misma carpeta donde se halle el ejecutable.

A.1.2 Modelo de objetos

Como se ha comentado anteriormente, el modelo de objetos define la estructura estática de los objetos del sistema y proporciona el entorno en el que situar el modelo funcional.

Se muestran a continuación el diagrama de clases, un diccionario de datos, una descripción de los métodos contenidos en el diagrama de clases y un glosario de términos.

Dentro de este subapartado se muestran únicamente las clases implementadas o modificadas en este proyecto, con el objetivo de que la relación entre ellas quede clara. Las clases cuyo nombre esté subrayado son las clases creadas para este proyecto; las clases con nombres no subrayados, son clases modificadas ya existentes en la versión anterior del *framework* TOYVision [TOYVision web]. En estas últimas clases solo se indican los atributos añadidos y los métodos añadidos o modificados.

A.1.2.1 Diagrama de clases

A continuación se muestra el diagrama de clases donde las flechas indican el sentido en que se hacen las llamadas a los distintos métodos.

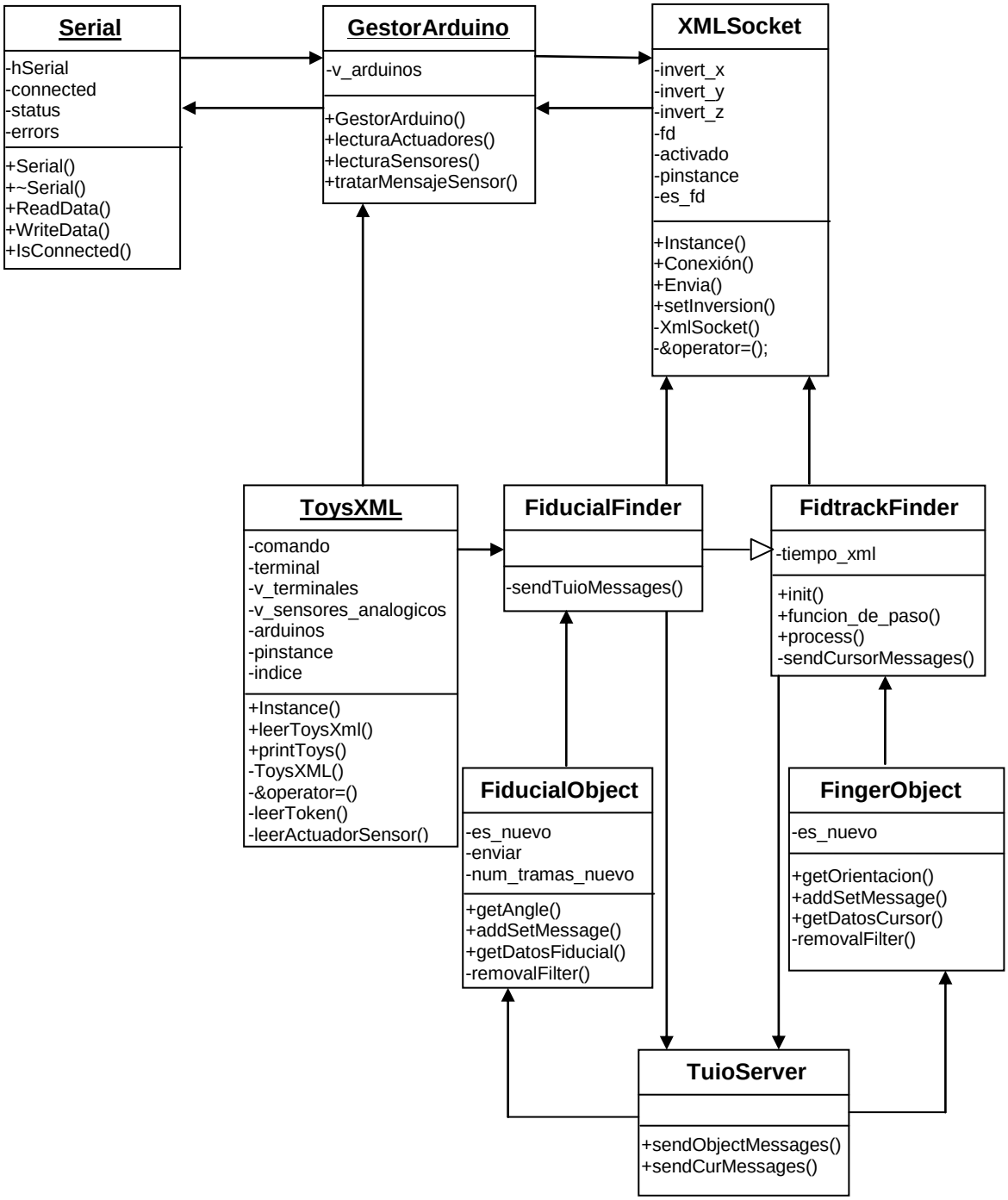


Figura A.1 Diagrama de clases

A.1.2.2 Diccionario de Datos

El diccionario de datos contiene la descripción de las clases presentadas en el diagrama anterior, haciendo referencia al alcance de cada una de ellas dentro del problema y cualquier suposición o restricción relativa a su uso.

Serial	Crea, controla y cierra la conexión creada con un puerto COM
GestorArduino	Gestiona toda la información intercambiada con la placa Arduino utilizando para ello la clase Serial
XMLSocket	Crea, controla y cierra la conexión hecha mediante un socket
ToysXML	Gestiona toda la información relativa a los juguetes que se vayan a utilizar, ya sean estos activos o pasivos
FiducialFinder	Gestiona todo lo relacionado con el envío de los eventos de los juguetes que lleven un fiducial adherido en su base
FidtrackFinder	Crea el *socket* y controla que la información a enviar relativa a los juguetes, *fingers* y *blobs* sea correcta. Además, gestiona el envío de los eventos de los *fingers* y los *blobs*
FiducialObject	Gestiona la información relativa a un juguete con un fiducial adherido a su base
FingerObject	Gestiona la información relativa a un *finger* o un *blob*
TuioServer	Gestiona el envío de los datos relativos a los eventos de juguetes, *fingers* y *blobs* a través del protocolo TUIO

Figura A.2 Diccionario de Datos de TOYVision

A.1.2.3 Definición de métodos

Serial

```
Serial(LPCSTR portName)
// inicializa la comunicacion Serial con el puerto COM 'portName'

~Serial();
// cierra la comunicación Serial. Una vez cerrada no se puede volver a
// crear la conexión, hay que cerrar el programa y volver a ejecutarlo

int ReadData(char *buffer, unsigned int nbChar);
// lee los datos contenidos en buffer provenientes de la conexión
// hecha. nbChar indica el número de bytes contenidos en buffer;
// si ese número es mayor que el número máximo legible de bytes,
// devuleve un número de bytes igual al máximo posible.
// Devuleve -1 si no ha leído nada

bool WriteData(char *buffer, unsigned int nbChar);
// envia a través de la comunicación Serial lo contenido en buffer,
// cuyo número de bytes es nbChar

bool IsConnected();
// comprueba si la conexión está hecha
```

GestorArduino

```
GestorArduino(void);
// obtiene el puerto COM (donde esta la placa Arduino conectada)
// con el que hacer la conexión, y activa los terminales de los
```

```
// sensores de la placa que vayan a ser utilizados.
// Además crea un thread que se encargue de hacer un escucha
// permanente de la placa Arduino para conseguir así la comunicación
// asíncrona.

lecturaActuadores (const char* buffer);
// lee el XML contenido en 'buffer' para poder analizarlo y enviar
// la orden correspondiente a la placa Arduino.

lecturaSensores(void * serialPort1)
// método que ejecuta el thread creado en el constructor y que
// se encarga de leer lo recibido desde la placa Arduino a través
// de 'serialPort1'

tratarMensajeSensor (char* buffer)
// traduce el mensaje recibido en lecturaSensores y contenido en
// 'buffer' para traducirlo y enviarlo a través del socket a la API.
```

XMLSocket

```
static XmlSocket* Instance();
// inicializa la varibale pinstance y se asegura de que solo
// haya una copia de la clase (la clase es de tipo singleton)

XmlSocket();
// cosntructor que inicializa los atributos

int Conexion();
// crea la conexión con el socket y se encarga de recibir los eventos
// que indiquen aceptación de la conexión, cierre de la conexión o
// datos recibidos

void Envia(char sendBuff[100000]);
// se encarga de enviar a través del socket el contenido de 'sendBuff'


void setInversion(int x, int y, int a);
// se guarda la informacion sobre si hay que invertir las coordenadas
// de las x, las y y/o el ángulo antes de enviarlas por socket. Esa
// información viene definida por el fichero de configuración
// reactivision.xml o indicada mediante la interfaz de reactivision

XmlSocket &operator= (const XmlSocket &);
// redefinición del operador '=' para poder hacer la asignación de los
// valores de un socket creado
```

ToysXML

```
static ToysXML* Instance();
// inicializa la varibale pinstance y es quien se asegura de que solo
// haya una copia de la clase (la clase es de tipo singleton)
```

```
ToysXML();
// constructor que inicializa los vectores v_terminales,
// v_sensores_analogicos y arduinos

void leerToysXml(const char *fileName);
// lee el fichero cuyo nombre lo indica 'fileName' y rellena el vector
// v_terminales con la informacion contenida en dicho fichero

void printToys();
// muestra el contenido de aquellos contenedores de los vectores
// v_terminales y v_sensores_analogicos que contengan informacion

ToysXML &operator= (const ToysXML &);
// redefinición del operador '=' para poder hacer la asignación de los
// valores de un de los vectores creados

void leerToken (TiXml::TiXmlElement *token);
// analiza un token guardando su informacion. El token puede ser un
// namedToken, un simpleToken o  un constraintToken

void leerActuadorSensor (TiXml::TiXmlElement *token,
                         TiXml::TiXmlElement *actuador, terminal
                         v_terminales[TOTAL_TERMINALES],
                         int es_sensor);
// analiza los actuadores o sensores correspondientes a un token
// guadando su informacion
```

FiducialFinder

```
void sendTuioMessages();
// prepara el mensaje XML a enviar a la API según el evento de un
// juguete ocurrido con un fiducial adherido a su base. Después,
// envía ese mensaje mediante el protocolo de comunicación elegido
```

FidtrackFinder

```
bool init(int w ,int h, int sb, int db);
// iniciliza los atributos y, en caso de que se haya elegido
// 'XMLSocket' como protocolo de comunicación en el fichero de
// configuración reactivision.xml, inicializa el socket y crea el
// thread indicándole el método a ejecutar para controlar el socket

void funcion_de_paso(void * socket);
// método  que ejecuta el thread creado en init. Se encarga de llamar
// al método conexión() de la calse XMLSocket

void process(unsigned char *src, unsigned char *dest, SDL_Surface
*display);
// trata la identificación hecha de los fiduciales, fingers y blobs
// por la cámara evitando enviar información falsa o duplicada

void sendCursorMessages();
// prepara los datos (en un  mensaje XML o en un mensaje TUIO) a
```

```
// enviar a la API según el evento de un
// finger o un blob ocurrido. Después, envía ese mensaje mediante el
// protocolo de comunicación elegido
```

FiducialObject

```
float getAngle() { return current.angle; }
// devuelve la orientacion de un juguete con un fiducial adherido a
// su base

std::string addSetMessage(TuioServer *tserver);
// devuelve una cadena de caracteres con la información de un juguete
// con un fiducial adherido a su base: número de sesión, identificador
// del fiducial, posición en el eje x, posición en el eje y y
// orientación. Este mensaje se envía si ha habido algún cambio de
// posición del juguete o si se ha verificado la detección de un
// juguete nuevo

std::string getDatosFiducial(TuioServer *tserver);
// obtiene los datos actuales de un juguete en concreto con un
// fiducial adherido a su base

bool removalFilter();
// elimina los datos de un juguete con un fiducial adherido a su base
// siempre que no haya aparecido en las últimas n tramas
```

FingerObject

```
float getOrientacion() {return current.orientacion; };
// devuelve la orientacion de un finger o un blob

std::string addSetMessage(TuioServer *tserver);
// devuelve una cadena de caracteres con la información de un finger
// o un blob: número de sesión, posición en el eje x, posición en el
// eje y y orientación. Este mensaje se envía si ha habido algún
// cambio de posición del finger/blob

std::string FingerObject::getDatosCursor(TuioServer *tserver);
// obtiene los datos actuales de un finger o blob en concreto

bool removalFilter();
// elimina los datos de un finger o blob siempre que no haya aparecido
// en las últimas n tramas
```

TuioServer

```
void sendObjMessages();
// envía mediante el protocolo 'TUIO' un mensaje relacionado con un
// evento relativo a un juguete con un fiducial adherido a su base.
// Este envío se hace únicamente si se ha elegido TUIO como protocolo
// de comunicación en el fichero de configuración reactivision.xml.
```

```
void sendCurMessages();
// envía mediante el protocolo 'TUIO' un mensaje relacionado con un
// evento relativo a un finger o a un blob.
// Este envío se hace únicamente si se ha elegido TUIO como protocolo
// de comunicación en el fichero de configuración reactivision.xml.
```

A.1.2.4 Glosario de términos

El glosario de términos contiene una lista de conceptos básicos necesarios para la correcta interpretación de algunos de los aspectos del proyecto.

Blob	Objeto al que no se le puede añadir un identificador. Está dirigido a aquellos objetos como manos, formas hechas con plastilina, un pincel…
Fiducial	Marcador formado por un conjunto de blancos y negros único, de forma que se le pueda diferenciar de los demás
Finger	Punto blanco como el que puede ser el creado con un dedo de un mano
Placa Arduino	Conversor analógico-digital con capacidad para tener un programa cargado según el actuar sobre actuadores u obtener información captada por sensores
Protocolo UDP	Protocolo de datagrama de usuario no orientado a conexión de la capa de transporte del modelo TCP/IP basado en el intercambio de datagramas. Este protocolo es muy simple ya que: - permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión (el propio datagrama incorpora suficiente información de direccionamiento en su cabecera) - no tiene confirmación ni control de flujo (los paquetes se pueden adelantar unos a otros) - no hay confirmación de entrega o recepción (no se puede saber si un paquete ha llegado correctamente)
Protocolo TCP	Protocolo de Control de Transmisión. Uno de los principales protocolos de la capa de transporte del modelo TCP/IP orientado a conexión, es decir, permite que dos máquinas que están comunicadas controlen el estado de la transmisión. Este protocolo de comunicación: - permite colocar los datagramas nuevamente en orden cuando vienen del protocolo IP - el monitoreo del flujo de los datos para así evitar la saturación de la red - que los datos se formen en segmentos de longitud variada para "entregarlos" al protocolo IP - multiplexar los datos, es decir, que la información que viene de diferentes fuentes (por ejemplo, aplicaciones) pueda circular simultáneamente en la misma línea - comenzar y finalizar la comunicación “amablemente”
Puerto COM	Puerto serial del ordenador a través del cual conectar un dispositivo
Socket	Método de comunicación entre un programa cliente y un programa servidor. Estos programas pueden o no estar ejecutándose en el mismo ordenador

Trama	Pasada del bucle principal del *framework* TOYVision en el que se reconocen los eventos ocurridos sobre la mesa, se traducen dichos eventos a datos tratables, se tratan y se envían a la API
TUIO	Protocolo de comunicación basado en un *socket* basado en un protocolo UDP unidireccional
XML	*Extensible Markup Language* (Lenguaje de marcas extensible). Formato usado para expresar información estructurada a través de etiquetas de la manera más abstracta y reutilizable posible. Una etiqueta consiste en una marca hecha en el documento, que señala una porción de este como un elemento

A.1.3 Modelo funcional

Como se ha dicho anteriormente, el modelo funcional se emplea para especificar el significado de las operaciones en el modelo de objetos. Para representar estas operaciones se utilizan los Diagramas de Flujos de Datos (DFD). Los DFD son grafos compuestos por nodos y arcos. Los nodos representan actores (producen/consumen datos), procesos (transforman datos) o almacenes de datos (elementos pasivos que guardan datos). Los arcos por el contrario, pueden ser valores de entrada/salida o flujos de datos (valores intermedios).

En la Figura A.3 se muestra el funcionamiento general de TOYVision mediante del DFD de nivel 0. En este nivel, el *framework* TOYVision recibe la imagen captada por la cámara y envía a la API la interpretación de dicha imagen.

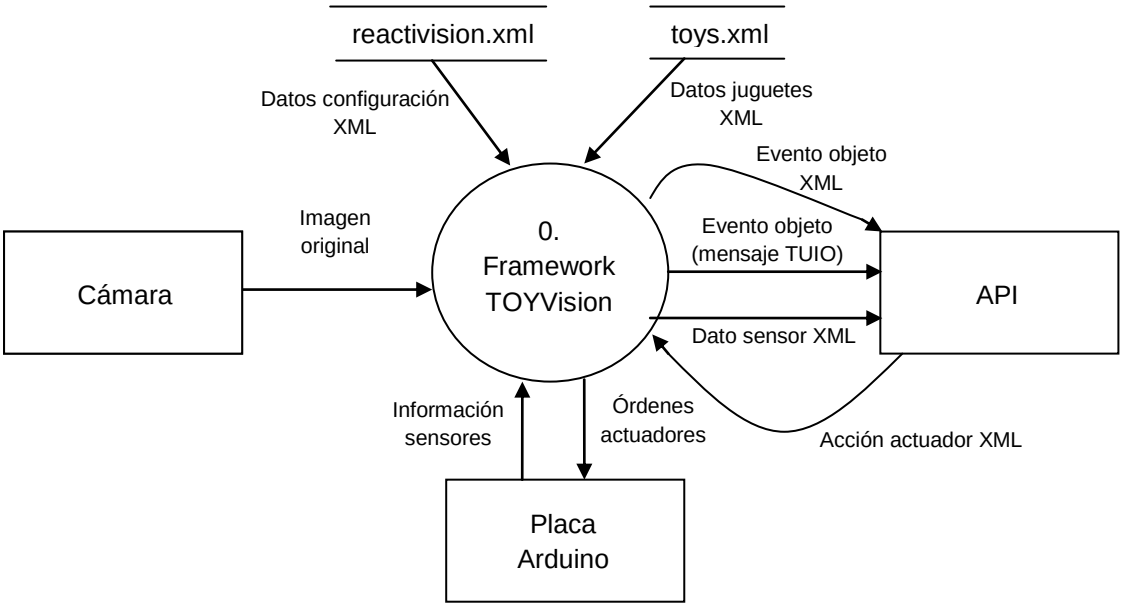


Figura A.3 DFD de nivel 0

En la Figura A.4 se muestra mediante el DFD de nivel 1 el funcionamiento general del *framework* TOYVision.

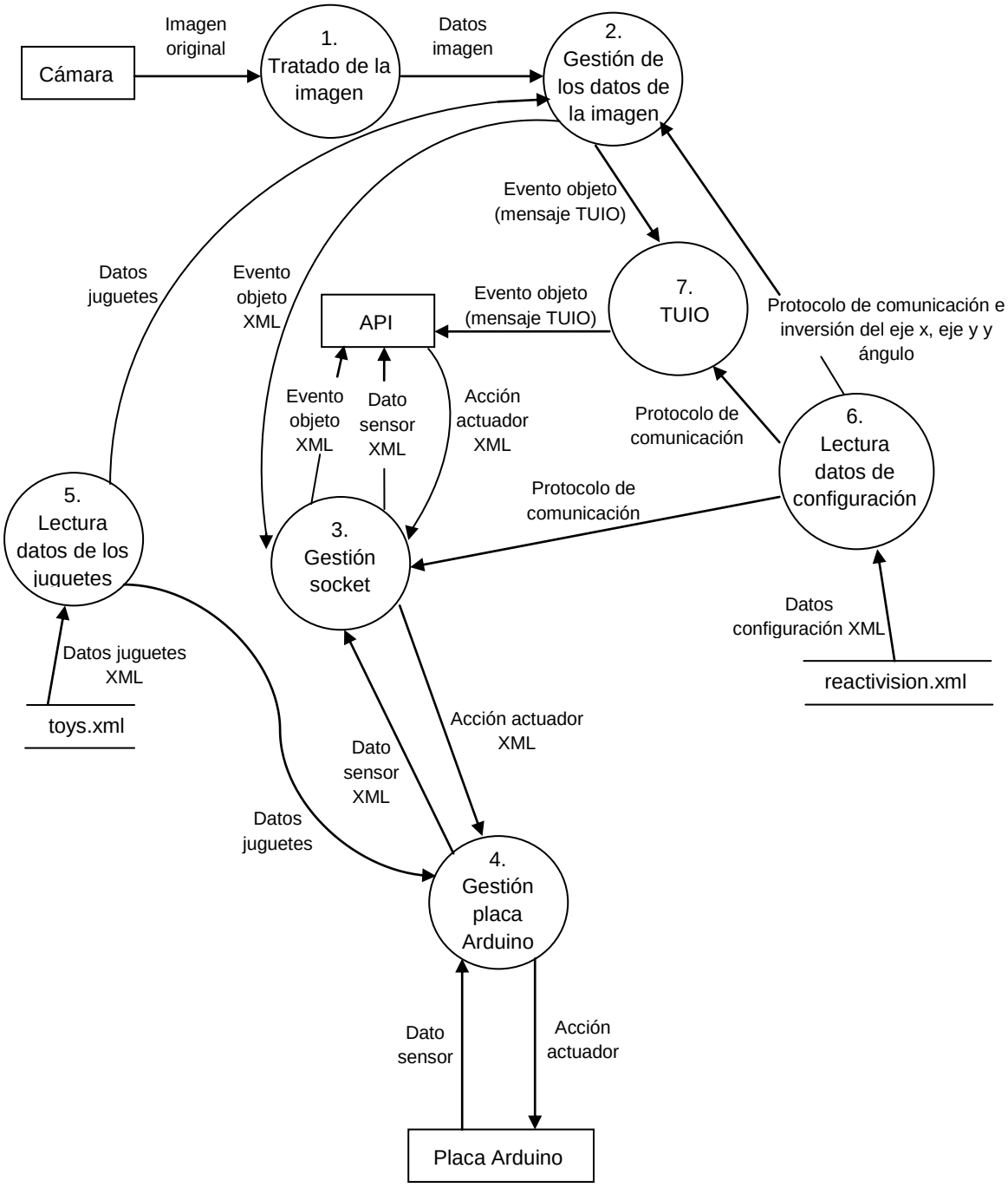


Figura A.4 DFD de nivel 1

En la Figura A.5 se muestra mediante el DFD de nivel 2 el funcionamiento del *framework* TOYVision, detallando esta vez parte de los procesos del nivel 1 del DFD.

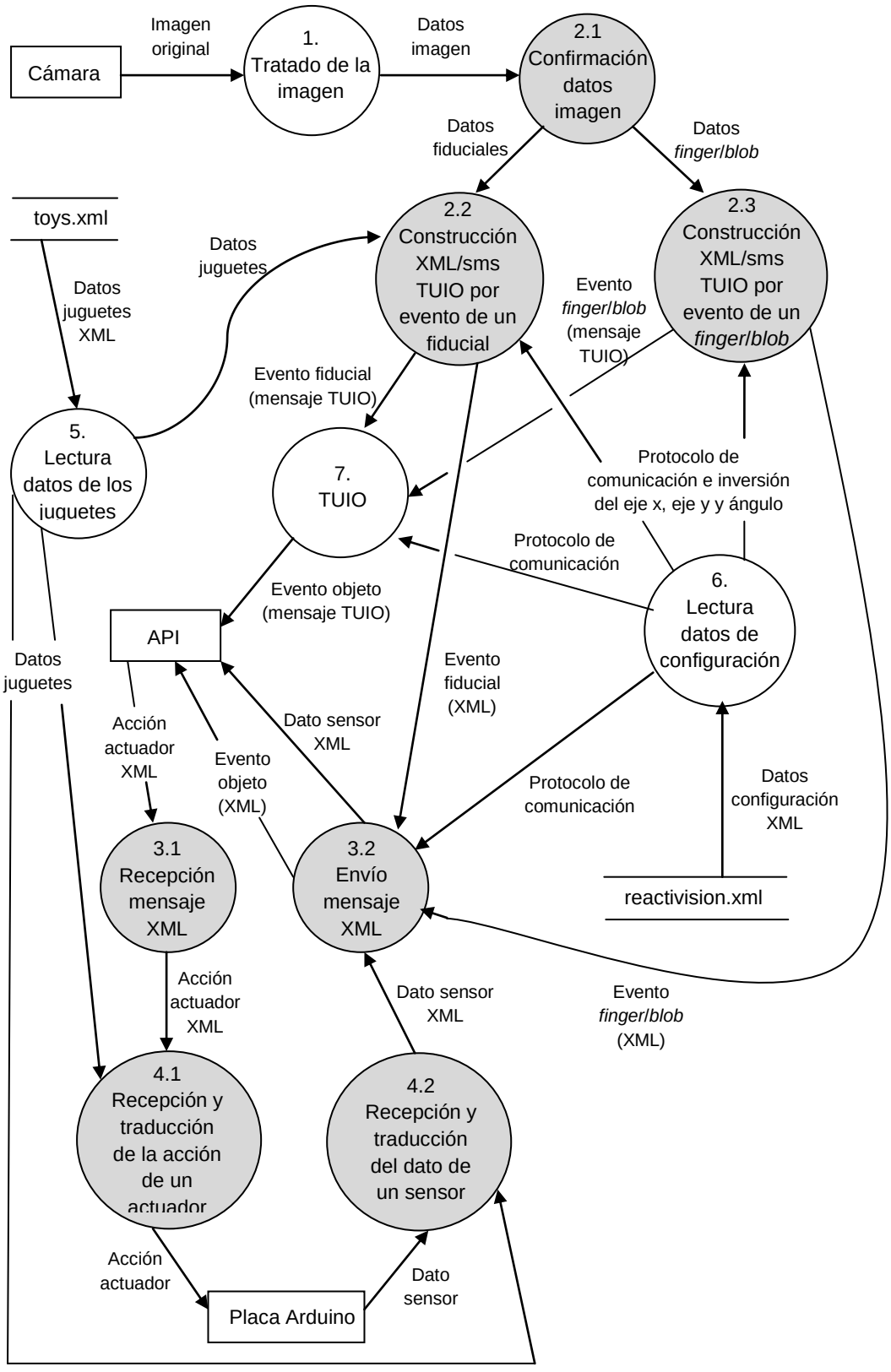


Figura A.5 DFD de nivel 2

A.2 Fichero de configuración *toys.xml*

El fichero de configuración *toys.xml* creado por un asistente gráfico almacena todos los datos relativos a los juguetes y a la placa Arduino. La razón de elegir XML como tipo de fichero es la facilidad de modificación que contiene.

Teniendo en cuenta el análisis hecho sobre la información relativa a la placa Arduino y a los juguetes que es necesario almacenar:

- Arduino
    - Identificador del Arduino.
    - Puerto COM del ordenador al que está conectado el Arduino.

- Juguetes
    - Nombre del juguete.
    - Identificador del fiducial de la base del juguete.
    - Dispositivos electrónicos asociados (si es el caso)
        - Nombre del dispositivo electrónico.
        - Terminal del Arduino al que el dispositivo electrónico está conectado.
        - Identificador del Arduino a cuyo terminal está conectado el dispositivo electrónico.
        - Tipo del dispositivo electrónico:
            - ‘non_continuous’ (servo de 180º).
            - ‘continuous’ (servo de 360º).
            - ‘switch’ (el valor relativo al dispositivo electrónico es ‘0’ o ‘1’).
            - ‘analog’ (el valor relativo al dispositivo electrónico está entre 0 y 1023, ambos inclusive).
        - Comandos:
            - Nombre del comando.
            - Tipo del comando:
                - ‘const’ (valor constante, viene definido en el ‘valor del comando’).
                - ‘var’ (valor variable, se obtiene en tiempo de ejecución).
            - Valor del comando: campo a tener en cuenta en caso de que el tipo de comando sea ‘const’ (constante).

Se presenta en la Figura A.6 la estructura del fichero hecha para obtener posteriormente un modo sencillo de almacenar en una estructura de datos toda esa información (la estructura de datos elegida en este caso se describe en el Capítulo 3 de la memoria).

```
<toys>
      <arduinos>
            <un_arduino id=" " port=" "/>
      </arduinos>
      <allNamedTokens>
            <namedToken name=" " fidID=" " copies="" shape=" " x=" "
      y=" " width=" " height=" ">
                  <actuator name=" " terminal=" " arduino=" "
                  type=" " icon=" ">
                              <command name=" " type=" " value=" "/>
                        </actuator>
                  <sensor name=" " terminal="" arduino=" " type=" "
            icon=" ">
                              <command name=" " type=" " value=" "/>
                        </sensor>
              </namedToken>
      </allNamedTokens>
      <allSimpleTokens>
            <simpleToken name=" " fidID=" " copies=" " width=" "
            height=" "/>
      </allSimpleTokens>
      <allConstraintTokens>
            <constraintToken name=" " fidID=" " copies=" " shape=" "
            x=" " y=" " width=" " height=" ">
                  <AssociativeArea name=" " radious=" " distance=" "
                  angle=" "/>
            </constraintToken>
      </allConstraintTokens>
</toys>
```

Figura A.6 Estructura del fichero toys.xml

Para una mejor comprensión de la estructura de este fichero, se presenta de una forma gráfica en la Figura A.7. Toda la información va contenida dentro de la etiqueta **toys**:

- **arduinos**: etiqueta que contiene las placas Arduino a utilizar. La información de cada placa está contenida en la etiqueta **un_arduino** cuyos atributos son:
    - **id**: identificador de la placa Arduino. Este identificador diferenciará una placa de las demás.
    - **port:** puerto COM al que la placa Arduino está conectada.

- **allNamedTokens**: contiene toda la información relativa a los juguetes con un fiducial en su base y los dispositivos electrónicos que tengan embebidos, si es el caso. La información de cada juguete está contenida en la etiqueta **namedToken** cuyos atributos son:
    - **name**: nombre del juguete elegido por el usuario.
    - **fidID**: identificador del fiducial gracias al cual se puede diferenciar al juguete de otros distintos (lo asigna el asistente gráfico).
    - **copies, shape, x, y, width, height**: datos utilizados por el asistente gráfico, no relevantes en este proyecto.

En el caso de que un juguete lleve al menos un dispositivo electrónico embebido, dentro de la etiqueta **namedToken** relativa al juguete, habrá una etiqueta **sensor** (indica que el dispositivo es un sensor) o **actuator** (indica que el dispositivo es un actuador) para cada uno de los dispositivos electrónicos. Los atributos para cada una de estas etiquetas son:

- **name**: nombre del dispositivo, elegido por el usuario.
- **terminal**: terminal de la placa Arduino al que conectar el dispositivo, impuesto por el asistente gráfico.
- **arduino**: identificador de la placa Arduino a cuyo terminal se hace referencia en el atributo anterior.
- **type**: tipo de dispositivo indicado por el usuario. Las posibilidades son:
  - switch: los valores a aplicar/leer pueden ser 0 o 1 (todos los dispositivos excepto los *servos*).
  - analog: los valores a aplicar/leer oscilan entre 0 y 1023 (todos los dispositivos excepto los *servos*).
  - continuous: *servo* continuo; según el valor aplicado el *servo* girará en un sentido y a una velocidad determinados.
  - non_continuous: *servo* no continuo; su posición puede estar dentro del rango de 0º a 180º, siendo este valor el que se le pasa para elegir la posición deseada.
- **icon**: dato utilizado por el asistente gráfico, no relevante en este proyecto.

Dentro de cada una de las etiquetas **actuator** o **sensor**, hay una o más etiquetas **command** cuyos atributos son:

- **name**: nombre del comando definido por el usuario.
- **type**: tipo del comando. Las posibilidades son:
  - const: el valor es constante, es decir, cuando se haga referencia a este comando, el valor a tener en cuenta será el indicado en el atributo **value**.
  - var: el valor es variable, es decir, cuando se haga referencia a este comando, el valor a tener en cuenta será el obtenido en tiempo de ejecución.
- **value**: valor del comando elegido por el usuario en el caso de que el tipo del comando sea ‘const’.

- **allSimpleTokens**: contiene toda la información relativa a las fichas con un fiducial simplificado en su base. La información de cada ficha está contenida en la etiqueta **simpleToken** cuyos atributos son:
  - **name**: nombre de la ficha elegido por el usuario.
  - **fidID**: identificador del fiducial simplificado gracias al cual se puede diferenciar a una ficha de otras distintas.
  - **copies, width, height**: datos utilizados por el asistente gráfico, no relevantes en este proyecto.

- **allConstraintTokens**: contiene toda la información relativa a los juguetes a los que se les puede poner/quitar o modificar el estado de una pieza y que

llevan un fiducial en su base. La información de cada juguete está contenida en la etiqueta **constraintToken** cuyos atributos son:

- **name**: nombre del juguete elegido por el usuario.
- **fidID**: identificador del fiducial gracias al cual se puede diferenciar al juguete de otros distintos.
- **copies, shape, x, y, width, height**: datos utilizados por el asistente gráfico, no relevantes en este proyecto.

Cada **constraintToken** lleva una etiqueta **asociativeArea** cuyos atributos son **name**, **radious**, **distance** y **angle**. No se detalla información sobre estos atributos ya que son datos utilizados por el asistente gráfico, no relevantes en este proyecto.

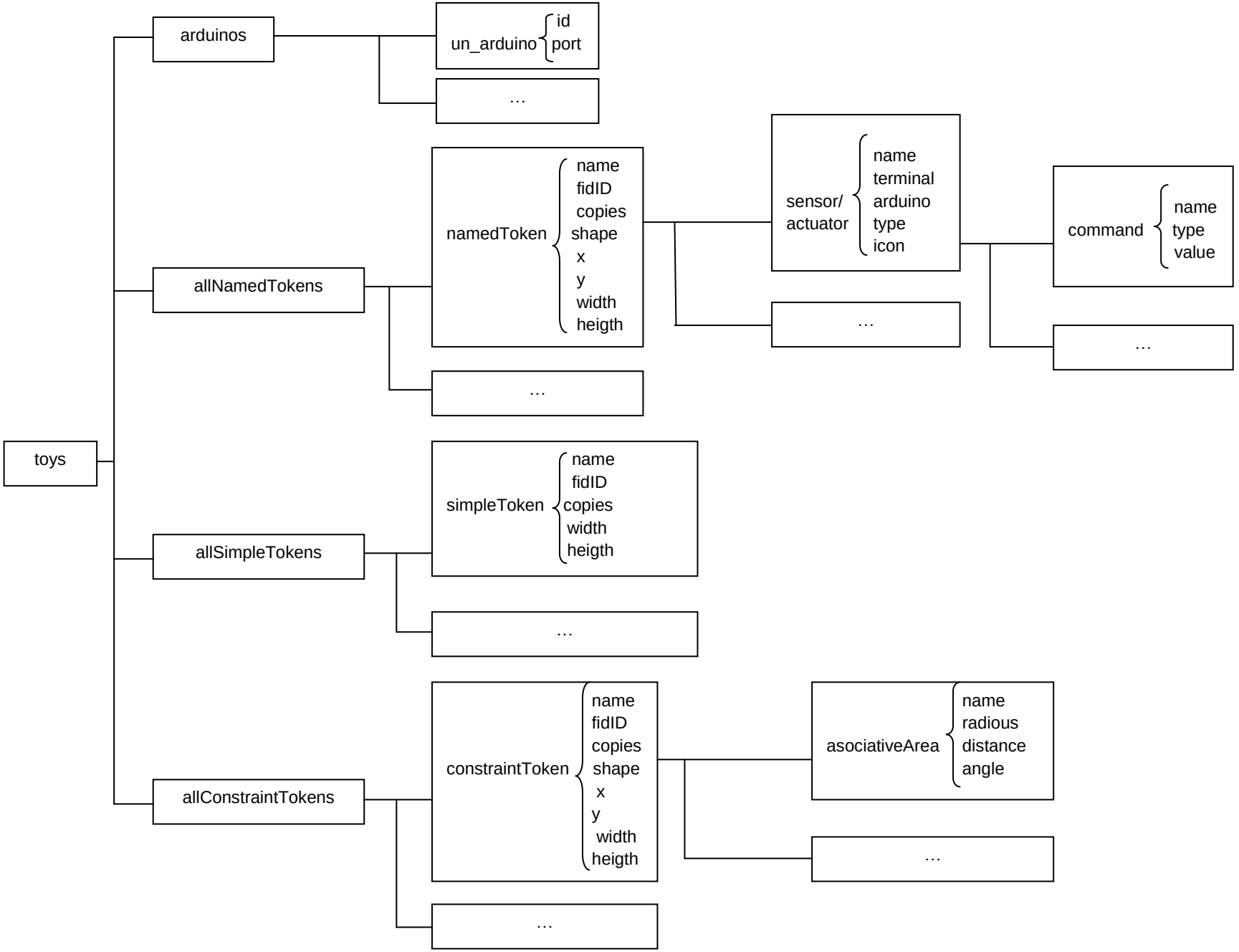


Figura A.7 Gráfico de la estructura del fichero toys.xml

A.3 Configuración de mensajes

Para hacer el intercambio de información entre el *framework* TOYVision y la API se utilizan mensajes XML. De esta manera se permite que la modificación sea sencilla (ya sea para incluir o eliminar algún dato).

Dependiendo de la información relevante en cada caso, el mensaje XML a enviar o recibir en el *framework* TOYVision varía. Los tres casos distintos en los que se envían o reciben mensajes XML son:

- **Eventos de los objetos:** para informar sobre un evento se debe diferenciar entre el tipo de evento y el objeto al que se hace referencia. Así pues, los distintos mensajes XML creados para cada evento y objeto son:

    - **Fiduciales:**
        - Añadir

        <fiducial_added id_session=" " name=" " pos_x=" " pos_y=" " orientation=" " />

        - Modificar

        <fiducial_updated id_session=" " name=" " pos_x=" " pos_y=" " orientation=" " />

        - Eliminar

        <fiducial_removed id_session=" " name=" " pos_x=" " pos_y=" " orientation=" " />

    - ***Fingers***
        - Añadir

        <cursor_added id_session="f " pos_x=" " pos_y="/>

        - Modificar

        <cursor_updated id_session="f " pos_x=" " pos_y=" "/>

        - Eliminar

        <cursor_removed id_session="f " pos_x=" " pos_y=" "/>

    - ***Blobs***
        - Añadir

        <blob_added id_session="b " pos_x=" " pos_y=" " orientation=" " area=" "/>

        - Modificar

        < blob _updated id_session="b " pos_x=" " pos_y=" " orientation=" " area=" "/>

        - Eliminar

        < blob _removed id_session="b " pos_x=" " pos_y=" " orientation=" " area=" "/>

- **Datos captados por los sensores:** cuando un sensor capta la variación de un dato, se transmite este nuevo dato a través de la placa Arduino hacia el *framework* TOYVision. Una vez recibido, el *framework* crea un mensaje XML para poder informar a la API del nuevo valor captado (para conocer la diferencia entre 'const' y 'var' véase el apartado anterior de este anexo):

- Si el comando del sensor que ha captado el nuevo valor está definido como **'const'**, se debe informar del nuevo estado:
  <sensor toyName=" " sensorName=" " status=" "/>

  donde:
  - **toyName**: indica el nombre del juguete.
  - **sensorName**: indica el nombre del sensor.
  - **status**: indica el estado con el que se corresponde el valor captado (si el valor no se corresponde con ningún estado definido, su valor será 'null').

- Si el comando del sensor que ha captado el nuevo valor está definido como **'var'**, se debe de informar del valor captado:
  <sensor toyName=" " sensorName=" " value=" "/>

  donde:
  - **toyName**: indica el nombre del juguete.
  - **sensorName**: indica el nombre del sensor.
  - **value**: indica el valor captado por el sensor.

- **Envío de órdenes a los actuadores:** para que la API pueda activar un actuador, debe hacerlo a través del *framework* TOYVision, para lo cual le envía un mensaje XML indicando el actuador y la forma sobre la que actuar:
  <command toy=" " actuator=" " status=" " value=" "/>

  donde:
  - **toy**: indica el nombre del juguete.
  - **actuator**: indica nombre del actuador.
  - **status**: indica el estado (comando) sobre el que actuar.
  - **value**: indica el valor a aplicar sobre el actuador (necesario en el caso de que el comando sea de tipo 'var').

A.4 Herramientas utilizadas

A continuación se enumeran las herramientas y tecnologías utilizadas a lo largo de todo este proyecto:

Sistemas operativos

- Microsoft Windows XP
- Windows 7

Tecnologías

- XML (eXtensible Markup Language)

Propósito general

- Navegador web: Mozilla Firefox
- Navegador web: Google Chrome
- Paint

Desarrollo y pruebas

- Entorno de desarrollo: Microsoft Visual Studio 2005
- Arduino 1.0.1

Documentación

- Microsoft Office Word 2007 y 2010
- Gantt Project 2.0.9

Dispositivos hardware

- Placa Arduino UNO

Anexo B. Arduino

En este anexo se presenta la placa Arduino UNO con cada una de sus partes detalladas, los dispositivos electrónicos con los que se ha trabajado en este proyecto y los circuitos a montar para hacer funcionar dichos dispositivos electrónicos.

B.1 Placa Arduino UNO

La Figura B.1 muestra el aspecto de una placa Arduino UNO:

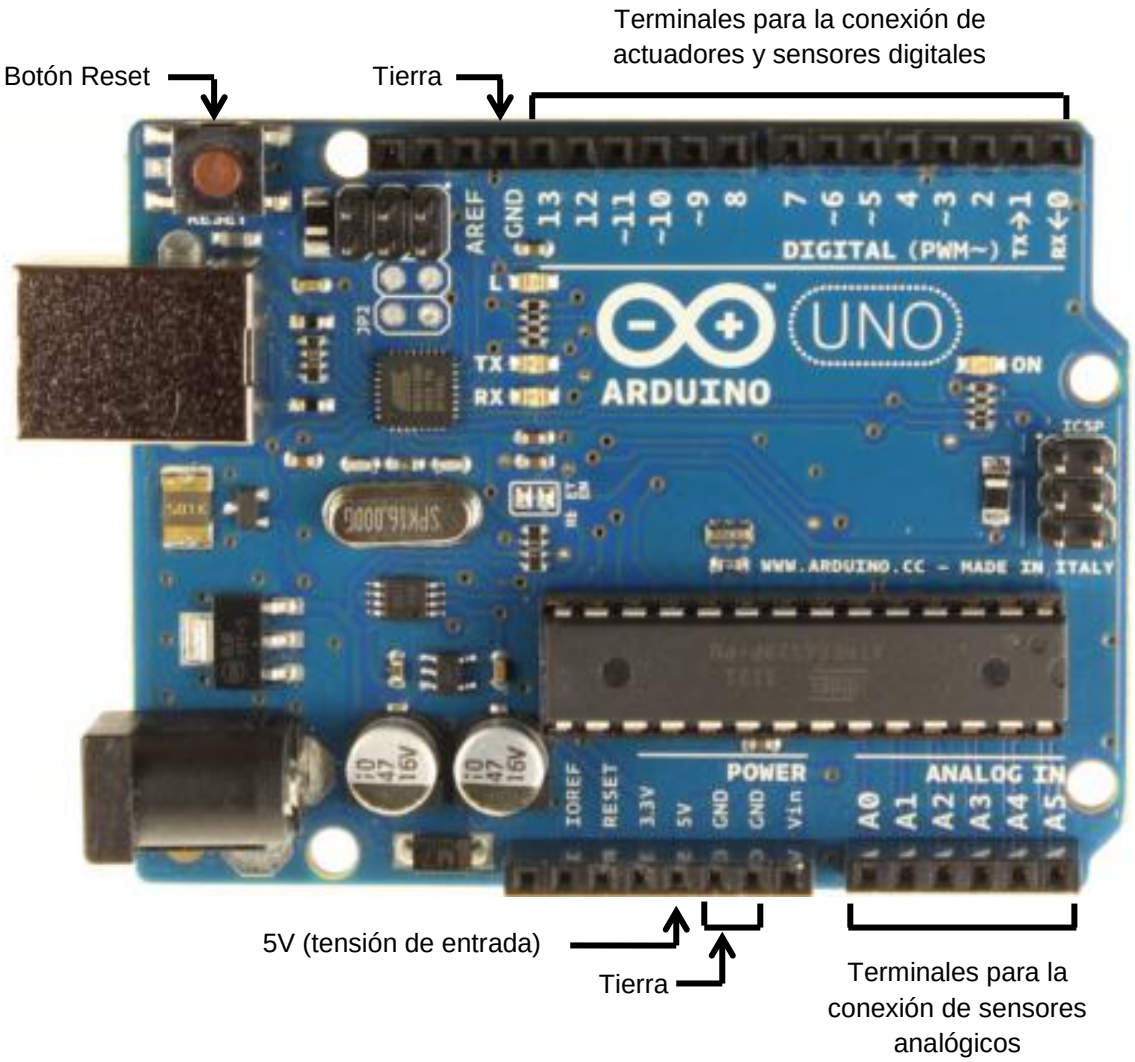


Figura B.1 Placa Arduino UNO

- **Botón reset:** sirve para inicializar la placa, de manera que se disponga a ejecutar la primera línea del código que esté cargado en ella. Es recomendable pulsar este botón al iniciar una ejecución para que las variables restablezcan sus valores iniciales.

- **GND (tierra):** para conectar un dispositivo electrónico con la placa es necesario montar un circuito. Uno de los extremos de dicho circuito debe ir siempre conectado a tierra para dar salida a los excesos de tensión.

- **5V (tensión de entrada):** al igual que todo circuito debe estar conectado a tierra, también debe estarlo a su fuente de alimentación. Estos 5V en ocasiones no son necesarios, ya que la tensión de entrada la puede dar el terminal a utilizar (es el caso de los *leds*, por ejemplo).

- **Terminales para la conexión de actuadores y sensores digitales:** aunque los terminales existentes son 14 (del 0 al 13), solo se pueden usar 12 de ellos (del 2 al 13) ya que los terminales 0 y 1 están reservados para uso interno de la placa. Los sensores conectados a estos terminales solo son capaces de devolver un '0' o un '1' (por ejemplo, un *led* o un interruptor). Dado que según el uso que se le dé al terminal el código a ejecutar debe ser uno u otro, con el objetivo de que no haya necesidad de cambiar el código cargado en la placa, los terminales están reservados de la siguiente manera:
    - Terminales 2, 3, 4 y 5: interruptores
    - Terminales 6, 7, 8 y 9: *leds* digitales (sus estados son encendido o apagado, no hay variación de intensidad)
    - Terminales 10, 11, 12 y 13: motores *servo*

- **Terminales para la conexión de sensores analógicos:** a diferencia de los terminales dirigidos a los actuadores y sensores digitales, se les puede dar uso a todos los terminales. A estos terminales se les pueden conectar sensores que devuelvan valores en un rango de 0 a 1023, tales como sensores de temperatura, de luz…

B.2 Dispositivos electrónicos utilizados

A continuación se presentan todos los dispositivos electrónicos (actuadores digitales, sensores digitales y sensores analógicos) con los que se ha trabajado, utilizando siempre el mismo código cargado en la placa Arduino.

Los **actuadores** probados son los siguientes:

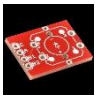
- Led:
    - Normal: emite luz blanca 
    - RGB: puede emitir cualquier color de luz 

- Infrarrojo: emite una luz infrarroja 

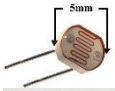
- Motor *servo*:
  - 180º: gira de 0 a 180º, según lo que se indique 
  - Continuo: gira continuamente en un determinado sentido y a una determinada velocidad 

- *Buzzer*: emite un pitido 

Los **sensores digitales** probados (devuelven ‘1’ (true) o ‘0’ (false)) son:

- Interruptor:
  - Normal: con pulsador 
  - De imán: se acciona al acercar un imán 
- Sensor táctil: se activa al tocarlo 

Los **sensores analógicos** probados son:

- Sensor de temperatura: mide la temperatura ambiente
- Sensor de luz: mide la intensidad de luz 

- Sensor de giro: mide la cantidad girada 

Dependiendo del dispositivo electrónico usado, el circuito a montar debe ser uno u otro. El circuito asociado a cada dispositivo electrónico se puede encontrar en el siguiente apartado.

B.3 Circuitos para los distintos dispositivos electrónicos

Dependiendo del actuador o sensor que se escoja, el circuito a montar es diferente. A continuación se muestran imágenes o instrucciones de cómo conectar cada uno de los dispositivos electrónicos comentados en el apartado anterior. Aunque se facilitan unos valores de resistencias a utilizar, se aconseja consultar al fabricante el mejor valor para cada dispositivo electrónico.

Actuadores

***Leds* (normales e infrarrojos):** la patilla corta va a tierra (GND) y la larga al terminal del que dependa el control. Entre la patilla larga y el terminal (cable azul) se debe conectar además, una resistencia de 220Ω (Figura B.2).

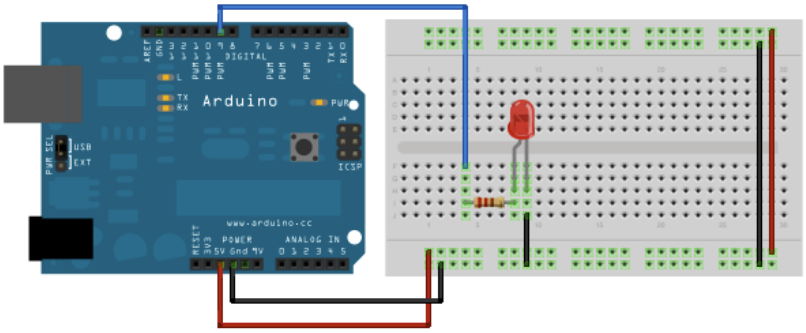


Figura B.2 Circuito para conectar un led normal o infrarrojo (2 patas) [Circuito led web]

***Leds* RGB:** la patilla más larga se conecta a tierra (GND) con un resistencia de 220Ω intercalada; cada una de las restantes tres patillas va a un terminal distinto que controla un color: la patilla de la izquierda se corresponde con el color rojo, la de la derecha con el azul y la que queda por asignar con el verde (Figura B.3).

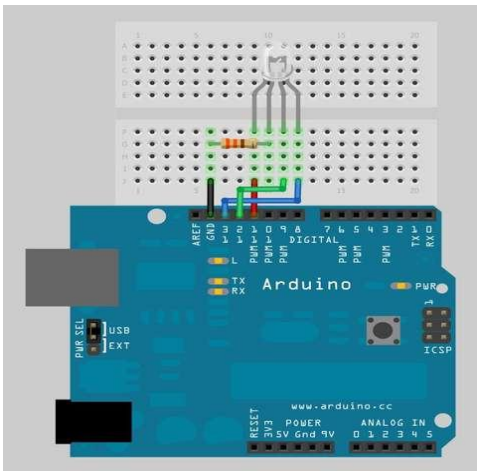


Figura B.3 Circuito para conectar un led RGB [Circuito led RGB web]

***Servos* (180º o continuo):** las conexiones se deben hacer tal y como se muestra en la Figura B.4: el cable rojo va a la fuente de alimentación (5V), el negro a tierra (GND) y el amarillo al terminal del que dependa el control del *servo*.

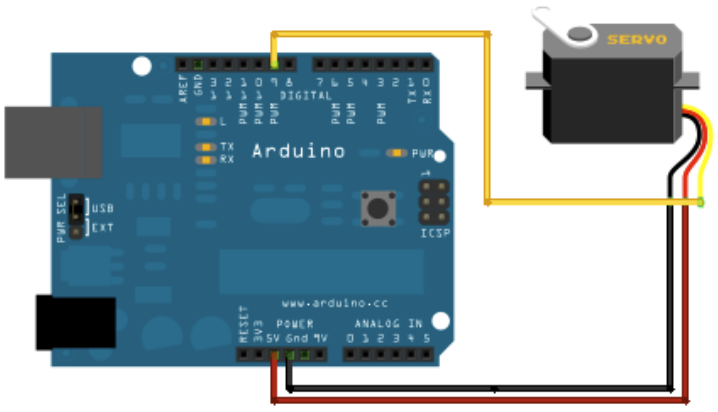


Figura B.4 Circuito para conectar un servo [Circuito servo web]

***Buzzer*:** respecto a la foto del *buzzer* mostrada en el apartado anterior, el cable rojo debe ir al terminal desde el que se quiera tener el control y el negro debe ir a tierra (GND) con una resistencia de 220Ω intercalada.

Sensores digitales

Interruptores con pulsador: el circuito a montar es el de la Figura B.5, donde el cable rojo va conectado a la fuente de alimentación (5V), el cable negro a tierra (GND) con una resistencia de 10KΩ intercalada, y el azul va al terminal desde el que se "lea" el estado del interruptor.

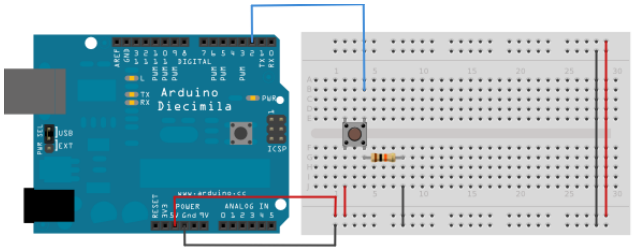


Figura B.5 Circuito para conectar un interruptor de botón [Circuito interruptor web]

Interruptores magnéticos: dado que solo tienen dos patillas (tal y como se observa en la foto asociada en el apartado anterior), una de ellas debe ir a la fuente de alimentación (5V) y la otra, tanto a tierra (GND) con una resistencia de 10KΩ intercalada, como al terminal desde el que se "lea" el estado del interruptor.

Sensor táctil: en el mismo sensor viene indicado a dónde debe ir conectada cada patilla:

- Patilla Vcc: a la fuente de alimentación (5V).
- Patilla GND: a tierra.

- Patilla Signal: al terminal desde el que se quiera “leer” si se está tocando el sensor o no.

Sensores analógicos

Sensores de temperatura y de luz: dado que tienen dos patillas (tal y como se observa en las imágenes asociadas en el apartado anterior), una debe ir conectada a la fuente de alimentación (5V) y la otra tanto a tierra (GND), con una resistencia intermedia de 1KΩ, como al terminal desde el que se quiera detectar el valor captado.

Sensor de giro: al igual que en el sensor táctil, en el mismo sensor viene indicado a dónde debe conectarse cada patilla:

- Patilla Vcc: a la fuente de alimentación (5V).
- Patilla GND: a tierra.
- Patilla Signal: al terminal desde el que se quiera “leer” si se está tocando el sensor o no.

Anexo C. Mejoras generales en el código del *framework* TOYVision

A lo largo del proyecto ha sido necesario revisar la estabilidad del código del *framework* TOYVision creado previamente, para obtener una mayor robustez. Esta inestabilidad era consecuencia de las sucesivas modificaciones que se habían ido realizando sobre el mismo código en proyectos anteriores [Nav11] [Enj10].

En los siguientes párrafos se presentan los problemas que se presentaron y que han sido solventados para conseguir la robustez mencionada.

A medida que se iban introduciendo nuevas funcionalidades en el código del *framework* TOYVision se comenzó a comprobar que este se volvía **inestable**. Por lo tanto, se tomó la decisión de volver a comenzar a introducir cambios a partir del código original de ReacTIVision, añadiendo todo lo relativo a este proyecto y a la detección de *blobs* [Nav11]. Al comenzar de nuevo, se hizo una gestión de la memoria mucho más cuidadosa, lo que le dio al *framework* TOYVision mayor estabilidad.

La mesa necesaria para NIKVision y presentada en el Capítulo 1 de la memoria, acepta distintas maneras de ser montada. Esto sin embargo, genera que a la hora detectar objetos el eje x, el eje y, y el sentido de la orientación necesarios para indicar la posición de los objetos, puedan quedar invertidos, por ejemplo, quedando la parte positiva del eje x a la izquierda. Por esta razón, tanto en el fichero de configuración *reactivision.xml* como en tiempo de ejecución, se puede elegir **invertir** estos tres datos (alguno o todos), de manera que cuando se vaya a indicar la posición de un objeto, se dé el dato correcto.

Al incluir el nuevo protocolo de comunicación, se perdió la capacidad de invertir los datos, por lo que se añadió en la clase *XMLSocket* el método encargado de comprobar si, en el fichero o en tiempo de ejecución, se habían invertido alguno o todos estos datos.

Debido a la simplicidad de los *simple token,* en ocasiones el *framework* TOYVision no los reconocía como tales, confundiéndolos con un *finger* y declarándolos en consecuencia, **fiduciales ilegales**. Por lo tanto, otra mejora a conseguir fue la de evitar esta confusión. La forma de obtener un buen resultado fue controlando que en un mismo lugar de la mesa no pudiera haber un *finger* y un *simple token* en instantes de tiempo inmediatos.

La **detección** de fiduciales, *fingers* y *blobs* no siempre se realiza correctamente, por lo que en muchas ocasiones la cámara deja de detectar durante un instante que uno de estos objetos está sobre la mesa.

Esta imperfección llevaba a que se simulara que un objeto se había eliminado de la superficie de la mesa, por lo que el *framework* TOYVision enviaba un mensaje de

“objeto eliminado”. Tras esa instantánea desaparición, el objeto volvía a aparecer, lo que simulaba la adición de un nuevo objeto sobre la mesa, es decir, el envío de un mensaje de “nuevo objeto”.

Por otra parte, en ciertas ocasiones se confundía un fiducial con otro distinto, por lo que cuando un fiducial aparentaba haber sido eliminado, al volver a aparecer como “nuevo”, no lo hacía como el mismo fiducial, sino como aquel que era parecido a él. Todas estas confusiones se traducían en una mala ejecución del juego.

Estos problemas se solventaron esperando un determinado tiempo para asegurar que efectivamente un objeto se había eliminado o añadido. Es decir, para enviar un mensaje de “nuevo objeto” u “objeto eliminado”, el objeto en cuestión tenía que haber estado sobre la mesa o haber estado desaparecido durante un determinado tiempo.

Al comenzar la ejecución del *framework* TOYVision, el usuario debe regular la apertura y ganancia de la cámara hasta unos niveles adecuados que eliminen el ruido en la imagen, es decir, inicialmente (y hasta que no se hace dicha regulación) aparece mucho ruido. Ese ruido se interpreta como objetos según la forma que tenga (*blobs* mayormente), lo que se traduce en una gran cantidad de información a enviar: continuos mensajes de “nuevo objeto”, “objeto modificado” y “objeto eliminado”.

Por lo tanto, si el ordenador utilizado no era muy potente, al predefinir el uso de *blobs* (en el fichero de configuración *reactivision.xml*), nada más comenzar la **ejecución**, esta se paraba y cerraba automáticamente imposibilitando su eso. Tras muchas pruebas de distintos tipos, se decidió hacer esta última mejora esperando 15 segundos (que es lo que se calculó que se tarda en calibrar la cámara) tanto para empezar a detectar *blobs* como para hacer el primer envío con la toda la información de la pantalla (indica la forma de los *blobs*). Es decir, se daban 15 segundos para configurar la cámara y asegurar que la ejecución no se parara por el exceso de información a enviar.

Anexo D. Gestión del Proyecto

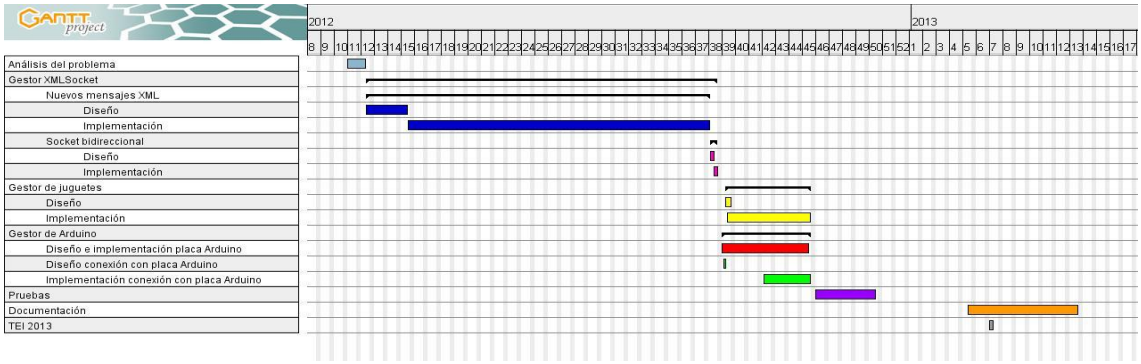


Figura D.1 Evolución temporal del proyecto

La Figura D.1 muestra la evolución temporal del proyecto mediante un diagrama de Gantt. A continuación se detallan cada una de sus partes indicando las fechas en las que están comprendidas:

Análisis del problema (12/02/2012 – 22/03/2012): en esta fase del proyecto (de color azul claro en el diagrama) se incluye no solo lo documentado en el Anexo A sino también el estudio previo que se tuvo que hacer de todo el código del *framework* TOYVision ya existente.

Gestor XMLSocket (22/03/2012 – 24/09/2012; 2-4hs/día): esta fase se corresponde con el Capítulo 4 de la memoria e incluye tanto el envío de los nuevos mensajes XML (de color azul oscuro en el diagrama) como el nuevo *socket* bidireccional (de color rosa en el diagrama). Es la fase que más tiempo abarca por dos razones: la primera es que durante toda esta fase se tuvo que combinar el desarrollo del presente proyecto con asignaturas de la titulación, de manera que el número de horas dedicado por día fue menor al habitual; la segunda razón es que durante esta fase se hicieron gran parte de las mejoras citadas en el Anexo C, ya que es la fase donde más problemas y errores se encontraron.

Gestor de juguetes (26/09/2012 – 12/11/2012): durante esta fase (de color amarillo en el diagrama) se hizo todo lo relacionado con lo explicado en el Capítulo 3 de la memoria contenida en este documento. Además, se incluye todo el tiempo invertido en el estudio de la librería *TinyXML*, utilizada para la lectura de ficheros de tipo XML.

Gestor de Arduino (24/09/2012 – 12/11/2012): esta fase se corresponde con el Capítulo 5 de la memoria y se realizó en paralelo con la fase del Gestor de juguetes, ya que los errores que iban a apareciendo en esta parte afectaban a la otra, y viceversa. Aquí se incluye tanto lo relacionado con la placa Arduino (de color rojo en el diagrama) como lo relacionado con la conexión del *framework* TOYVision con dicha placa (de color verde en el diagrama). Dentro de la placa Arduino, las fases de diseño e implementación están unidas en una sola ya que, a causa de ser algo totalmente

nuevo, la manera de aprender a utilizarlo fue ir haciendo distintas pruebas con distintos dispositivos electrónicos.

Pruebas (12/11/2012 – 14/12/2012 y 09/02/2013 – 11/02/2013): en el primer intervalo de fechas de esta fase (de color morado en el diagrama) se hicieron todas las pruebas necesarias utilizando el ejemplo mostrado en el Capítulo 6 de esta memoria. Durante esta fase se encontraron varios errores a los que se les fue dando solución hasta conseguir una ejecución perfecta. Al finalizar este intervalo de fechas se dejó durante un tiempo el proyecto a un lado para poder preparar el último examen de la carrera.
El segundo intervalo de fechas (de color gris en el diagrama) se corresponde con el congreso internacional hecho en Barcelona donde se hizo un *workshop* gracias a este proyecto (tema detallado en el Capítulo 7 de la memoria). En este congreso también se hicieron pruebas ya que hubo que enfrentarse a casos a los que antes no se había tenido oportunidad de enfrentarse.

Documentación (31/01/2013 – 31/03/2013): esta fase (de color naranja en el diagrama) comprende la redacción del documento aquí presentado. Aunque se fueron tomando notas a lo largo de todo el proyecto, fue en esta fase donde se pusieron en común para explicar todo el proceso que ha supuesto este proyecto.

Se presenta a continuación y mediante la Figura D.2 el tiempo total dedicado a cada una de las fases del proyecto.

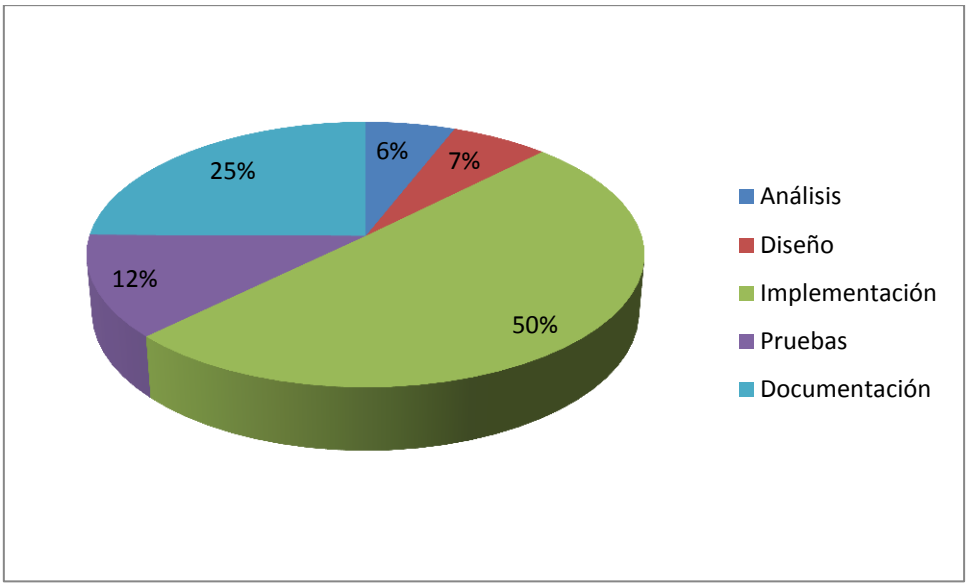


Figura D.2 Porcentaje dedicado a cada fase del proyecto

El número de horas invertidas en cada fase en concreto es:

Fase	Nº de horas
Análisis	**32** (6%)
Diseño	**35** (7%)
Gestor de XMLSocket Gestor de Juguetes Gestor de Arduino	24 5 6
Implementación	**271** (50%)
Gestor de XMLSocket Gestor de Juguetes Gestor de Arduino	120 71 80
Pruebas	**67** (12%)
Documentación	**134** (25%)
Número total de horas	**539**

Índice de figuras

Referencias bibliográficas

[Amicus web] Página oficial de Amicus http://www.myamicus.co.uk/
(último acceso 31/03/2013)

[Arduino 1.0.1 web] Software para Arduino de la página oficial de Arduino
http://arduino.cc/en/Main/Software (último acceso 31/03/2013)

[Arduino Serial web] Código de la clase Serial de la página oficial de Arduino
http://playground.arduino.cc/Interfacing/CPPWindows
(último acceso 31/03/2013)

[Arduino web] Página oficial de Arduino http://arduino.cc/ (último acceso 31/03/2013)

[Circuito interruptor web] Circuito para utilizar un interruptor con la placa Arduino http://arduino.cc/en/Tutorial/Button (último acceso 31/03/2013)

[Circuito *led* RGB web] Circuito para utilizar un *led* RGB con la placa Arduino
http://2.bp.blogspot.com/_yVs2qD6ICNU/TQU2MLRCmoI/AAAAAAAAAKI/W-Hpp75NM8g/s1600/Wire-It-Up-Common-Cathode.jpg (último acceso 31/03/2013)

[Circuito *led* web] Circuito para utilizar un *led* normal o infrarrojo con la placa Arduino
http://arduino.cc/en/Tutorial/Fade (último acceso 31/03/2013)

[Circuito *servo* web] Circuito para utilizar un *servo* con la placa Arduino
http://arduino.cc/en/Tutorial/Sweep (último acceso 31/03/2013)

[Demo juego web] Juego demostrativo del juego del dragón
http://vimeo.com/59922197 (último acceso 31/03/2013)

[Enj10] Enjuanes, Alejandro. 2010. Adaptación de algoritmos de reconocimiento visual para la implementación de juegos para niños basados en dispositivos Tabletop. PFC.

[KBBC05] Kaltenbrunner. Martin. Bovermann, Till. Bencina, Ross. Costanza, Enrico. 2005. TUIO: A Protocol for Table-Top Tangible User Interfaces. Proc. of the 6th International Workshop pn Gesture in Human-Computer Interaction and Simulation.

[Lenguaje Arduino web] Lenguaje de programación para la placa Arduino
http://arduino.cc/es/Reference/HomePage (último acceso 31/03/2013)

[MVS2005 web] Entorno de desarrollo Microsoft Visual Studio 2005
https://partner.microsoft.com/spain/40013160 (último acceso 31/03/2013)

[Nav11] Navarro, Guillermo. 2011. Reconocimiento visual de juguetes en una mesa de interacción tangible. PFC.

[RBP96] Rumbaugh, James. Blaha, Michael. Premerlani, William. Eddy, Frededick. Lorensen, William. Modelado y diseño orientado a objetos. Metodología OMT. Prentice Hall, España, 1996.

[ReacTIVision web] Código fuente del ReacTIVision original http://reactivision.sourceforge.net/ (último acceso 31/03/2013)

[TEI 2013 web] Página oficial del TEI 2013 http://www.tei-conf.org/13/ (último acceso 31/03/2013)

[TinyCLR] Página oficial de TinyCLR http://www.tinyclr.com/ (último acceso 31/03/2013)

[TinyXML tutorial] Tutorial de la clase TinyXML http://www.grinninglizard.com/tinyxmldocs/tutorial0.html (último acceso 31/03/2013)

[TOYVision web] Página oficial de TOYVision www.toyvision.org (último acceso 31/03/2013)

[TUIO web] Página oficial del protocolo TUIO http://www.tuio.org/ (último acceso 31/03/2013)

[TUIOAS3 web] API en AS3 para el protocolo TUIO http://www.tuio.org/?flash (último acceso 31/03/2013)

[W3C] Página oficial de W3C http://www.w3c.es/ (último acceso 31/03/2013)

[Workshop TEI 2013] Página oficial del *workshop* del TEI 2013 http://webdiis.unizar.es/~jmarco/?p=432&lang=es (último acceso 31/03/2013)